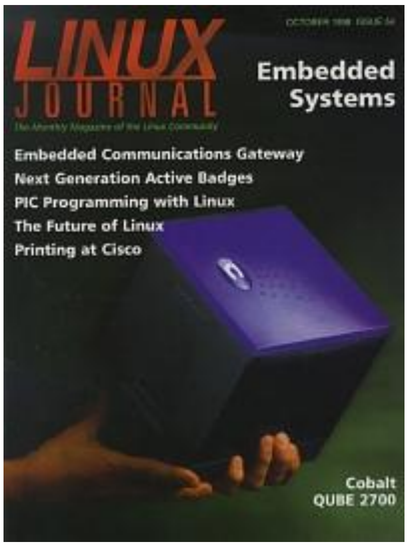
Advanced search

Linux Journal Issue #54/October 1998



Archive Index

Advanced search

Linux in an Embedded Communications Gateway

Greg Herlein

Issue #54, October 1998

This article describes a communications gateway system, why Linux was chosen for the implementation and why Linux is an excellent choice for similar gateways.

In 1996, Pacific Gas & Electric Company (PG&E) began a project to develop a commercial-grade advanced sensor to monitor electric lines. The system was simple in concept: use lots of small cheap units that hang on the electric lines, each consisting of a few sensor elements, an embedded CPU, a spread-spectrum radio and a battery. The sensor units would monitor for certain line conditions; if those conditions occurred, the sensor would conduct measurements, and then radio the information to a “master” radio unit. That master unit would, in turn, forward the information through a gateway to a server on the corporate WAN.

Server software would analyze the arriving data and, if warning conditions existed, would alert the electrical system operators. Other software on the WAN would be capable of monitoring and performing remote configuration of the wireless sensor network. This software would use a proprietary wireless networking protocol to accomplish the control and monitoring—in effect, tunneling this protocol within TCP/IP out through the gateway system and back.

Figure 1. Gateway Schematic

The project was on an aggressive timetable: to put a working demonstration in the field before winter storms began in the fall of 1996. Luckily, the underlying sensor technology had already been developed by the PG&E Research and Development Lab. PG&E partnered with an outside vendor to develop, test and bring to market a wireless network of sensors based on this sensor technology. The total system was to consist of three major parts: the sensor network, the client software to use the data collected by the sensors and a gateway providing a reliable link. Linux was chosen as the operating system for the gateway.

System Description

The goal of the project was to develop and test sensor technology and its associated wireless network. R&D had pioneered the actual sensor design and had conducted field tests of prototype sensors. To move the technology from prototype to product, we had to design for manufacturing—not something with which a utility company has experience. In addition, developing and testing a wireless spread-spectrum radio network was beyond our abilities. Therefore, we partnered with a commercial vendor to help develop and produce the system.

The vendor was to integrate our sensor designs with their radio and to implement the radio network. We would provide full technical assistance and handle all data collection and analysis for the initial trials. A fairly substantial software development effort would also be necessary. The client-side display software was based on other, related software development efforts, so that part of the project had a head start. I was a member of the project development team in R&D and was responsible for the actual development of the gateway software.

The wireless sensor network used spread-spectrum radios. These unlicensed radios are relatively immune to interference, but require a clear line of sight between transmitter and receiver. The sensors were designed to hang from power lines. To provide a line of sight to those sensors and good geographical coverage from a single master radio, the radio site must be above the level of the power lines. Thus, the master radio must be on a hilltop or mounted on a tower to be effective. Either solution poses problems in getting power and a WAN connection to interface with the master radio. To add to the challenge, a given site might require more than one master radio, depending on geography. To “look” over a knoll or around a building, we might have to add a second master radio. Even in those circumstances, we wanted to use only one gateway system; therefore, each gateway must be able to serve several master radios.

Figure 2. The author (foreground) and Julian Riccomini installing the gateway system.

The field trial site for the project was chosen for its interesting electrical distribution circuits, topography typical of our electrical service area and a relatively accessible site for a master radio. The site was several hours away from our office location at the top of a large, steep hill. The road up the hill was treacherous even when conditions were dry; in wet winter months, reaching our equipment location at the top of the hill would be very difficult. Given the difficulties of physically reaching the gateway system, it was imperative that it be reliable and fault-tolerant. The site already had a backup electrical generator protecting other radio equipment located on the hilltop, which significantly simplified our power requirements for the gateway system and master radio. A small UPS added ride-through capability to protect the systems during voltage sags, and while waiting for the backup generator to come on-line in the event of an outage.

The master radio was custom-developed by the wireless vendor, but its link to the gateway was limited to a single serial communications port. No embedded Ethernet or other networking interface would be included in the radio. The gateway system had all networking responsibility. To further complicate the matter, radio antenna requirements forced the master radio to be located outside, high up on a mast. The computer system was to be located inside a nearby building. The serial communications cable between the master radio and the computer system would have to extend about 50 feet. This eliminated standard RS-232 communications, since at those cable lengths we would never attain the design goal of 56 baud. Also, normal RS-232 communications would not allow us to have multiple master radios per gateway without adding more serial ports. To solve the problem, we chose to use RS-485 two-wire differential communications, which gave us distance, high baud rates, excellent noise immunity and the ability to expand the link to multiple drops (supporting multiple radios) if needed.

Figure 3. Area overlooked by the gateway system.

The gateway would require additional functionality for the field trials. To properly evaluate the data gathered from the sensors, data had to be recorded somewhere. The system specifications called for data to be delivered in real time to a central server that would record it, perform data analysis and, if necessary, send an alarm to the electrical system operators. Any failure of the links from the gateway to the WAN or a WAN outage would break the entire process and risk data loss. In fact, the times when the most interesting data is collected—during storms—is the very time links are most likely to fail. Obviously, the gateway must have redundant links and the ability to log sensor data locally, should all links fail. The hilltop had a direct line of sight to a local PG&E office and a few spare telephone wire pairs. For the primary link, we used a commercial spread-spectrum, point-to-point radio bridge. (Similar radios were already in wide use in the company as WAN links.)

Our backup link was ISDN. We chose to use a commercial bridge to provide automatic switching between the radio and the ISDN links. While we could have added that capability in our software, we chose to buy it “off-the-shelf”, believing the extra cost was worth the reduction in development time. To provide additional redundancy, we added a standard dial-up line. This would provide an alternate path for retrieving data should a WAN outage occur, severing the link on the network side. As a complete backup, the gateway would also log all network traffic from the master radio. Those logs would be kept on the system's hard drive, since we expected to repair any link failure before recording many megabytes of data. Of course, we still might have to physically go to the gateway and perform a manual download, but we wouldn't lose data.

As discussed above, the master radio communicated to the gateway across an RS-485 link. This link used a simple protocol (similar to HDLC, high-level data link control) to packetize the proprietary radio network protocol and allow for multiple master radios when required. The gateway would take the data stream from the RS-485 link, encapsulate it into a TCP/IP socket connection and flow it onto the WAN. Data flowing in the other direction would go back out the RS-485 link to the master radio. On the WAN, a simple database server would be at the other end point of the protocol tunnel. The software used by the electrical system operators would interact only with the server. This modularization was key to dividing the work among the disparate groups participating in the development process.

While designing the system, we always had an eye on scalability. Each WAN-based server needed to be capable of supporting multiple wireless network gateways. Successful completion of the project would mean rolling out several hundred wireless networks, leading to large database requirements. In addition, the server software would need to run on hardware not specifically purchased for this project. All PG&E divisions had large Sun servers in place for support of other engineering functions. From both a financial and a space perspective, the idea of buying and installing new servers just for the wireless sensor network was rejected. Our new server software had to be portable across major UNIX platforms—at a minimum, SunOS and Solaris. Recall that the server was to be an end point for the protocol tunnel carrying the proprietary radio network data. The code we wrote for the server had to be portable to the gateway as well; writing two versions was not only silly, but also out of the question given our short development timetable.

All in all, the bridging system and its associated components had quite a list of requirements, including the following:

- Speak RS-485 on one side and Ethernet on the other.
- Support tunneling the radio network protocol across a serial communications channel and within TCP/IP.
- Support a primary, a backup and a dial-up link.
- Support automatic data logging locally if those links failed.

- Support the ability to extract logs over the network without disrupting the gateway operations when links were restored and without disrupting normal operations.
- Be as reliable as possible to operate in a remote, occasionally inaccessible location.

The server software must be portable across various flavors of UNIX and reuse as much of the gateway code as possible. To make matters more exciting, the entire system had to be built, tested and installed in the field in about six months.

For much of the system, we chose to use off-the-shelf technology in order to minimize development time and maximize reliability. However, to our knowledge, this type of project had never been done before, and most of the special functionality would have to be written by us. Without a stable operating system with a reasonable set of features and tools, we'd never be able to meet the deadline.

Why Linux?

Linux was immediately considered as an option for the project, due to positive experiences on an earlier project using Linux. However, the earlier project did data translations only in a batch environment in a normal office setup. While that earlier software was a major project, it was very different from a mission-essential “embedded” system. The gateway would be remote and unattended with no recourse to human intervention if something went wrong. We were certain Linux could handle the load, but prudence demanded the evaluation of alternatives. The gateway was essential to the entire project: without it, we'd have no performance metrics on how well the radio network of sensors performed. We needed solid reasons to justify our choice of Linux for the project.

As always, there were strong proponents for using a Microsoft OS. We rejected that approach immediately. Our experience with Windows NT indicated it still had some stability problems (despite being far more stable than other Windows platforms). While it is perhaps a useful desktop and/or server OS, we felt that NT was unsuitable for remote systems, particularly those without a display or a keyboard. We were also unhappy with the remote administration capabilities of NT. Perhaps most importantly, however, the system was to operate in an environment where it had to boot and run its software automatically if the watchdog timer ever reset the system. We had little faith in NT's ability to boot, correct disk problems and then run properly in this scenario. NT (and all other Microsoft Windows operating systems) were rejected for these reasons.

Various DOS-based solutions were also considered. We had serial port-handling libraries for DOS which we had used successfully in the past for solving the serial communications problems. Micro-Controller Operating System (*mu*C/OS), a freeware real-time OS available for many different microprocessors, was evaluated as well. *mu*C/OS had similarly good serial port-handling functions. Both were judged to be stable and reliable for an embedded application. However, one of the core system requirements was to allow log and data file extraction without disturbing the gateway operation. DOS would require some tricky programming to allow that option due to the uncertainty of some system calls, and we felt *mu*C/OS would have trouble with it under high traffic loads. Portability was another issue—the protocol tunnel code had to be easily portable to UNIX. Even if we could have located an affordable, reliable, TCP/IP stack for either DOS or *mu*C/OS, we had serious doubts about our ability to keep the source code portable. We removed both of those OS choices from the list.

We had a lot of choices among UNIX-like operating systems: QNX, Linux, FreeBSD, Solaris x86 or SCO. Our exposure to the latter two left us with serious concerns about their performance. None of the systems we had seen running those operating systems were particularly fast or responsive, even under fairly light loads. We also expected to port the gateway computer hardware platform to a single-board computer (SBC), and we wondered if either of those operating systems could run successfully on such hardware. We were certainly doubtful about the availability of RS-485 and watchdog timer drivers for those systems. We saw little value in choosing FreeBSD over Linux, since all our experience was with Linux.

So, our choices were reduced to QNX and Linux. QNX is a stable, multi-tasking, scalable, reliable, real-time OS that can easily run on SBCs. It has serial port/ RS-485 and watchdog timer support and full networking support, including TCP/IP. QNX also enjoys a reputation for good technical support. In short, a good initial fit was found between QNX and the project's needs. However, there were also pitfalls. QNX did not support **gcc** (we heard a port was in progress). The only compiler available was the one provided by QNX, based on the Watcom C integrated development environment. While it's probably a good compiler, it meant learning a new tool, and that translated to time lost in an already short development cycle. We were already using and familiar with gcc and the other fine GNU tools. Given the short project “time to market”, any time lost from learning the new environment was considered detrimental.

That same philosophy extended to QNX in general. As a sophisticated real-time, micro-kernel OS, a learning curve is involved to use it well. We asked ourselves if we actually needed a real-time OS with the extra complexity that would come with it. We also wondered about portability issues. Having no experience with

QNX, we were quite concerned that we would end up having to write the code twice: once for QNX and once for our server platform. At the very least, we were afraid we'd have to spend extra time adding conditional compiler directives to make the code work in both environments. In contrast, using Linux practically assured identical code. The licensing costs of QNX were substantial as well, in particular, when the full TCP/IP capabilities on every network node were added. In contrast, a single $50 Red Hat Linux CD-ROM represented a significant cost savings.

In the end, low cost, familiarity and availability of GNU tools tipped the scales in favor of Linux. It met all the project's key requirements at a tangible cost benefit which was hard to ignore. In summary, Linux provided:

- A stable, robust, multi-user OS
- Solid support for TCP/IP networking and serial communications
- Ability to run on various single-board and host computers
- Familiar, commonly used development tools with familiar system libraries—no time lost learning other tools and systems
- Code that was completely reusable on other UNIX platforms to implement the servers for the system

As an added advantage, if we ever went into production and began building hundreds or thousands of gateways, Linux would save us a significant amount of money because there are no license fees to use it.

Results

The gateway system was in place for more than a year with no significant downtime and no loss of sensor data. At one point, the system had 179 days of uptime—and the only reason it needed to be rebooted was as a result of troubleshooting a problem with the network bridge equipment. Linux has proved remarkably stable and effective, and human intervention has not been needed.

The system we installed in the field is a bit different than we originally envisioned. We had some trouble with the single-board computer we chose, but those problems were related to the on-board RS-485 hardware, not Linux. It looked grim for a few days as we fought communications problems and pored over the serial-driver source code to try to find a fix. Another major plus Linux has over other OS choices is the full availability of the source code. We had not expected to need it, but it certainly proved valuable. The developer of much of the Linux serial driver code, Theodore Ts'o, personally answered questions by e-mail—within hours in one case. While we certainly did not expect that response every time, we've never gotten that level of support from any commercial vendor.

Figure 4. Mary Ilyin and Julian Riccomini work on the master radio.

While resolving the hardware problems on the single-board computer, we shifted development work to an old 386-40MHz motherboard we had lying around. We swapped in the hard drive, added a network card, a watchdog timer card, and an RS-485 port card and immediately had a working gateway computer. The single-board computer was a 486-100, but we discovered Linux to be so efficient with minimal hardware, a faster CPU was not needed. We never did resolve the strange hardware problem on the single-board computer. The 386 system worked so well that we stayed with it and actually used it for the field trials. It worked perfectly. The incredible range of hardware that Linux supports became another solid plus that helped us implement a working system.

The server code also ran reliably. We tested and demonstrated the software on Sun servers running both SunOS and Solaris; however, the workhorse for the field system was a Pentium PC running Linux. The code base for the two systems was identical—the only difference was a few lines in the project Makefile. Our objective of portability was made trivial by Linux's adherence to open standards.

So, What Happened to the Project?

The wireless sensor network was a moderate success, although some technical problems did arise. However, during the middle of the project, changes in California law introduced more competitiveness into the state's electric utility industry. The sensor project had been started by PG&E's Department of Research and Development. One immediate change in the state's laws was that R&D funds would be administered differently. As a result, the company abolished its R&D department and transferred the project to another department to finish. The initial phase of the project is now wrapping up, but what steps will be taken next is still unknown.

A Hypothetical Approach to the Next Step

Linux has matured since this project began, and it's worth taking the time to examine how recent Linux developments might benefit the next stage of gateway design. I would like to emphasize that the following comments are hypothetical—if I were to attempt upgrading and modernizing the gateway system, this is the approach I would take.

The next stage would require deploying multiple radio networks and gateways to serve large geographical areas, and changes could be made to simplify and improve operations. Gateways in non-testing situations would not require as much logging of radio network traffic, so the hard drive, the only moving part in the system, could be removed.

Paul Moody has written an excellent new Mini-HOWTO on embedded Linux (http://users.bigpond.com/paulmoody/). His document makes adding a flash disk as simple as following a recipe.

For our field trial, we specifically chose to use external, discrete network bridging equipment to link to the corporate WAN. We felt it would be fast to implement and more reliable than something we wrote ourselves. It wasn't quite plug-and-play (configuring the bridge proved troublesome), but it was relatively quick to set up. However, the bridging equipment was the only source of problems the gateway experienced in nearly eighteen months of service, so I'm no longer convinced of the superiority of proprietary network equipment. In contrast, the custom gateway software performed flawlessly.

The June 1998 issue of *Linux Journal* had an excellent article on the use of the Sangoma WANPIPE S508 router card (http://www.sangoma.com/). In addition, Usenet reports from users include raves about the Spellcaster ISDN cards (http://www.spellcast.com/). Using those cards in a future system would integrate all the gateway functionality into one box (with the exception of a radio for a point-to-point link, if such a link was needed). The resulting system would be about half the size, approximately two-thirds cheaper, use half the power, and in all likelihood, be even more reliable than the present system.

Figure 5. Mary Ilyin inspects an antenna on top of the building that hosts the gateway.

Conclusions

This project is one of many examples of Linux being successfully used in the commercial marketplace and in mission-critical commercial communication applications. In fact, Linux may have a place in such applications where perhaps no other OS can compete.

Linux is exceptionally well-suited for communications functions. It's fast, stable, reliable, has built-in TCP/IP networking, and it uses common, well-known development tools. Commercial versions of those tools are now available from Cygnus Solutions (http://www.cygnus.com/). Using loadable modules makes trimming the system down to a minimum kernel fairly easy to do, and the whole operating system is Open Source, so there's no limit to the customization possible.

Linux runs on many different processors, opening a wide range of target platforms—yet applications developed for it can easily be ported to other UNIX platforms, as required. The number of Linux ports to small, cheap processors is growing—Linux even runs on the Palm Pilot. Commercial technical support for Linux is available from several companies. Linux developers are getting easier to find as the popularity of the OS increases, and as more universities use Linux in their computer science labs. In addition, and perhaps most important to a commercial venture, is the lack of any license fee to use Linux. That translates into a significant cost savings for commercial products. With its technical and financial advantages, Linux may be the best option for many embedded communications systems.

The coming years will see a huge array of new commercial communications services being offered. PCS systems, new applications for wireless networks, low-earth orbit satellite networks, new utility SCADA systems, home and small office networks—all of these systems will need to be interconnected in various ways, and embedded communications systems will be built to serve these functions. These systems will range from cable-modem/video-on-demand controllers mounted high on telephone poles to small black boxes locked away in phone company central offices to small black boxes that sit behind your PC. I won't be surprised if many of these next generation systems run Linux.

Acknowledgements

Greg Herlein no longer works for Pacific Gas and Electric Company. Since building the system described here, he has gone on to adventures as network administrator on a major oceanographic research ship. He now owns and runs Herlein Engineering, a software development and consulting company that specializes in Linux, networking and communications and is based in San Francisco. You can find out more at http://www.herlein.com/ or reach him by e-mail at greg@herlein.com.

Archive Index Issue Table of Contents

Advanced search

PIC Programming with Linux

Brian C. Lane

Issue #54, October 1998

Mr. Lane has written a program called picprg to enable you to easily program a PIC microcontroller.

The vast majority of the computers in the world do not run Windows. While this is good news for Linux enthusiasts, the bad news is that they don't run Linux or much of any kind of operating system at all. These are the computers running your televisions, VCRs, cell phones, pagers and marine radios. They go by cryptic names like MC68HC05, 87C51, or PIC16C84 and are manufactured by companies like Motorola, Philips and Microchip.

Microcontrollers are the workhorse computers of the world. They do the repetitive tasks that require little or no human intervention and most of them will not even blink when the “millennium” bug hits their larger, faster cousins. They power up, do their job and power down again using very little power and defiantly not requiring a heat sink and a fan.

One of these little wonders is the PIC16C84 from Microchip. This is an 18-pin processor with 1KB of electrically erasable/programmable read-only memory (EEPROM), 36 bytes of SRAM and 13 input/output lines; it can operate at speeds ranging from DC (0Hz) to 10MHz.

The PIC16C84 is an excellent introduction to embedded processors and assembly language. The RISC instruction set has only 35 commands (op-codes) to learn, and the cost is under $8 for one. You can build a PIC programmer for under $20 in parts, or you can buy one pre-built and pre-tested over the Internet. Prototype boards are also available that need only a processor; they already have the clock crystal and programmer header, as well as a small prototyping area for adding to the circuits (usually a couple of LEDs for your first project).

This low cost for development doesn't mean that the PIC cannot be used for serious work. Several of my projects include an interface between the PC and the Dallas Semiconductor 1-wire bus, and a wired remote control that uses the Sony Control-L protocol to control a camcorder. In the most recent Circuit Cellar Ink contest, one of the winners implemented the PPP and TFTP protocol using an 8-pin PIC12C672.

Because of the ease of designing and building a PIC programmer to attach to a parallel port, dozens of designs are available, all using different pins on the parallel port. Some use inverters on all the control lines, and others use inverters on only some of the lines. My program **picprg** can handle all of these, as long as they use the standard five control lines. With all of these variations, the software to drive the programmer needs to be easily configurable.

Another feature of these devices is the ability to design a programming header into the circuit so that the processor can be programmed without removing it from the device it is attached to. This facilitates software work in the field, allowing technicians to easily service and upgrade the software.

When I first started using the PIC16C84, a compiler was already available for Linux, but no Linux software ran the HOPCO programmer that I use. An easy way to solve this problem would have been to get the DOS software included with the programmer to run under DOSEMU. Since I never seem to pick the easy way, I decided to write a native Linux PIC programmer. I decided on a full-screen ncurses interface, which would run on a VT console or an xterm as long as the **TERM** environment variable is set to **xterm-color**.

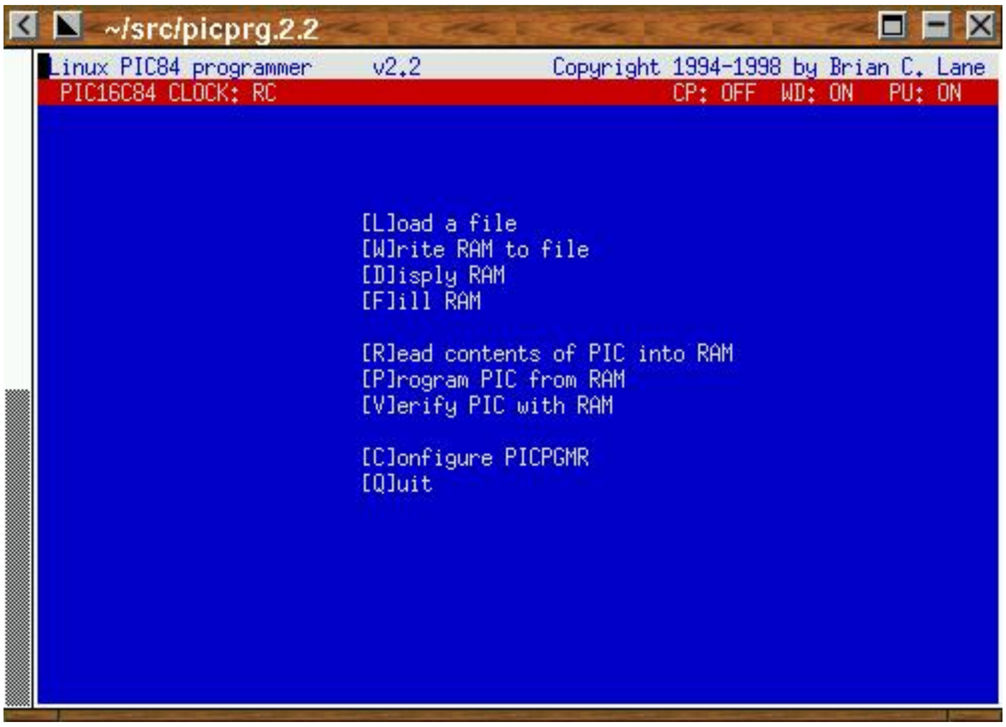
My picprg program allows you to program the PIC microcontroller, read previously programmed PICs, verify a PIC against the program in memory, and view the program in hexadecimal. It also features a versatile configuration screen, which makes it a snap to use with the wide variety of PIC programmers available.

Compiling picprg

Compiling picprg is easy: you just type **make** in the source directory and a binary called picprg is generated. The only dependency for picprg that may cause problems is the ncurses library. You must have v1.9.9e or later installed for it to work. All of the Linux distributions that I know of include ncurses by default, so you should be set. If you want to install it as suid root in /usr/local/bin, then type **make install**; otherwise, you will have to move it to your preferred final location.

picprg must be run as root, since it requires low-level access to the /dev/lp device that isn't available to normal users, even with write access enabled. You can either run it as root or install it as suid root, so that it can run as the root user. Remember that any program running suid root is a potential security risk.

The first time picprg is started, you must pass it the number of the printer port (/dev/lpX) to which you have attached the programmer. I have my modified HOPCO programmer attached to /dev/lp2, so I run **picprg -p2** to start it for the first time. You will see a nice blue screen (I'm still addicted to the color scheme of my Atari 800) as shown in Figure 1.



```
Linux PIC84 programmer      v2.2          Copyright 1994-1998 by Brian C. Lane
 PIC16C84 CLOCK: RC                                  CP: OFF  WD: ON   PU: ON

                        [L]oad a file
                        [W]rite RAM to file
                        [D]isply RAM
                        [F]ill RAM

                        [R]ead contents of PIC into RAM
                        [P]rogram PIC from RAM
                        [V]erify PIC with RAM

                        [C]onfigure PICPGMR
                        [Q]uit
```

Figure 1. picprg Start-up Screen

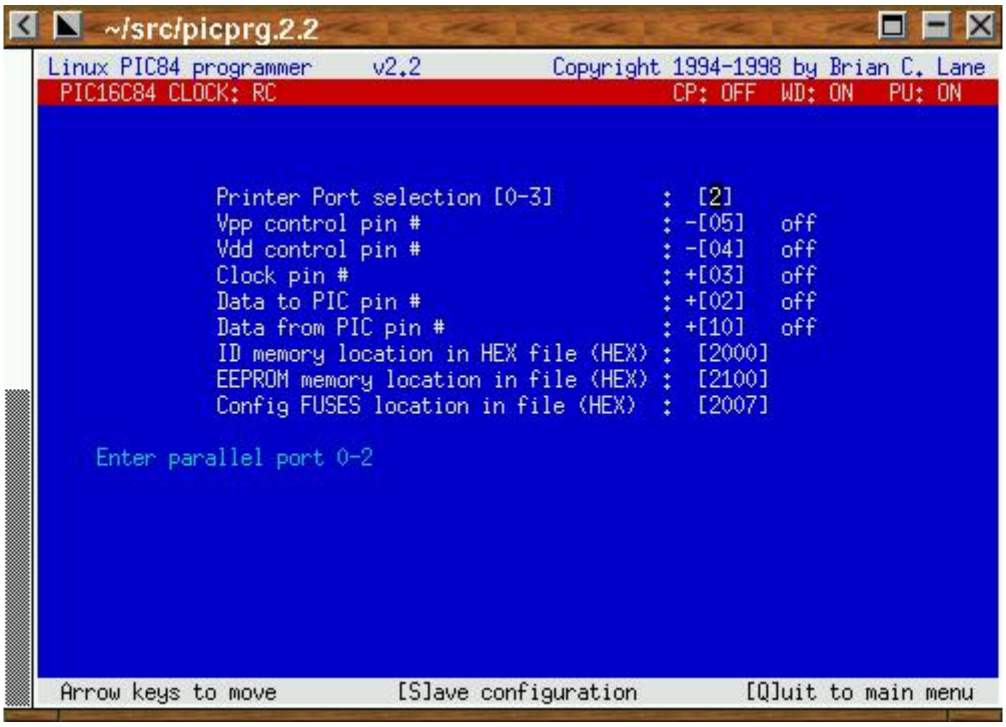
The main menu is self-explanatory. Pick option **C** to get the configuration menu. Use the arrow keys to navigate the list of configuration options, and a short help message will be displayed for each selection.

Configuration

I am using a programmer from HOPCO that has been re-wired to use the same parallel port connections as a David Tait-style PIC programmer. The connections for my programmer are as follows:

```
   Vpp control        pin 5  inverted logic
   Vdd control        pin 4  inverted logic
   Clock              pin 3  normal   logic
   Data to PIC        pin 2  normal   logic
   Data from PIC      pin 10 normal   logic
```

The connections for your programmer will most likely be different, depending on the pins it uses to connect to the parallel port and the transistor logic used to implement the on/off control. The configuration menu (see Figure 2) allows you to easily specify which pins are being used and the on/off logic used to control them. Press **+** and **-** to switch the logic used for control of each pin.



```
~/src/picprg.2.2
Linux PIC84 programmer      v2.2          Copyright 1994-1998 by Brian C. Lane
 PIC16C84 CLOCK: RC                             CP: OFF   WD: ON    PU: ON


               Printer Port selection [0-3]           :  [2]
               Vpp control pin #                      : -[05]   off
               Vdd control pin #                      : -[04]   off
               Clock pin #                            : +[03]   off
               Data to PIC pin #                      : +[02]   off
               Data from PIC pin #                    : +[10]   off
               ID memory location in HEX file (HEX)   :  [2000]
               EEPROM memory location in file (HEX)   :  [2100]
               Config FUSES location in file (HEX)    :  [2007]

          Enter parallel port 0-2

 Arrow keys to move          [S]ave configuration          [Q]uit to main menu
```

Figure 2. Configuration Menu

After you have entered the correct pin number for each function, test it to ensure that the PIC being programmed is seeing the correct logic level. The configuration menu is used for this, too. When one of the control lines is selected using the arrow keys, the **O** and **F** keys can be used to turn that control line On and Off. This on and off state is defined from the perspective of picprg, taking into account the polarity you specified when setting up the pins.

Testing

Now, get out your trusty voltmeter or whatever you are going to use for detecting power (it should be able to handle at least 13V). Connect the meter's GND to pin 5 on the programming socket. Then select Vpp in the configuration menu and hit **O** and **F** to turn it on and off. You should see pin 4 going from 0V to 13V or so (depends on the programmer). Repeat this for each of the pins listed below:

```
Vpp      pin 4   0 and approximately 12 volts
Vdd      pin 14  0 and 5 volts
Clock    pin 12  0 and 5 volts
Data Out pin 13  0 and 5 volts
```

The state of the **Data In** pin on the configuration display should be the same as the state of the **Data Out** pin. When **Data Out** is 1, the **Data In** line should be 1 as well.

Once you have all the voltages swinging the right direction, press **S** to save the configuration to ~/.picprgrc; type **picprg**. You are now ready to start using your PIC programmer.

ID and FUSE Configuration

When reading the object file, the ID and FUSE data are taken from the memory locations specified in the configuration menu. Microchip defines this to be **0x2000**, but some assemblers place this data in different locations. The configuration menu allows you to specify where in the loaded file the ID and FUSE data will be found. If you are using the **picasm** assembler, the default values for ID, FUSE and EEPROM location do not need to be changed. They will work correctly right out of the box.

EEPROM Configuration

The PIC16C84 has 64 bytes of internal EEPROM data that can be programmed with data like a unique serial number or configuration parameters. This data can be included in the loaded file at the memory location specified by the configuration menu. This address is usually set to **0x2100**.

Supported File Types

picprg uses the Intel HEX 16 format, which is supported by most assemblers. (I recommend picasm by Timo Rossi.) The FUSE and EEPROM locations can be defined in the configuration menu if your assembler does not put them in the standard locations. **picasm** also supports the Intel HEX 8 format, and picprg detects this file format automatically. Just type in the file name, and the program will figure out in which format the file was saved.

Loading an Object File

When an object file is loaded, it is stored in the internal memory buffer. The clock type and fuse states are displayed on the second line of the display while the file is in memory. The clock type and fuse states are determined by looking at the data stored in the FUSE memory location as defined in the configuration menu.

Reading a PIC

Make sure the programmer is connected and the processor is plugged in correctly. Press the **R** key from the main menu and the data from the processor will be read into the internal buffer; its clock type and fuse states will be displayed on the second line of the display. As it is reading, picprg will display every 16th address, just to let you know it is working correctly.

Programming a PIC

With the programmer connected and the processor plugged in the right direction, press **P** to start the programming process. The data in RAM will be written to the PIC and the third line will display “Programming PIC”. If any errors are found, they will be displayed with the address, the value read from the PIC during the failed verify and the value expected. Every 16th address it programs will be displayed, assuring you that it is still working.

From this point, it's up to you to learn how to write programs for the PIC. Many useful sites which can help you on your way are on the Internet; David Tait's list of PIC resources is the best starting point.

Resources

Brian Lane lives with his wife Denise in Olalla, Washington. He spends his days developing embedded software and his nights writing Linux code. He can be contacted at nexus@tatoosh.com or http://www.tatoosh.com/nexus/.

Active Badges—The Next Generation

Igor Bokun

Krzysztof Zielinski

Issue #54, October 1998

Implementing a software location system as a Linux embedded application results in a robust, efficient and inexpensive system.

An increased interest in location-aware computer systems has occurred recently. This class of computer systems is able to react to location changes of people, equipment or resources. These systems could create a base for applications such as a “follow me” computer environment that supports user mobility around a building. The main part of such a system forms a location system.

Glossary

This article describes our experiences during the implementation of a selected component of a new ABng (Active Badges Next Generation) software component for the location system called Active Badge. The ABng software has been implemented as a CORBA application using the distributed object paradigm.

The component of the ABng software that we implemented is called the poller: it acts as a communication engine between Active Badge Systems' sensors and the rest of the location system. Its efficiency and reliability have a substantial influence on system performance. Despite this strategic role, the poller is a self-contained, plug-and-play hardware component, easy to install and inexpensive.

The main purpose of this paper is to show that the usual trade-off problems between robustness, efficiency and low price can be solved by implementation of the poller as a Linux-embedded application. All consequences of this design decision have been analyzed in detail to evaluate its advantages and disadvantages.

The Active Badges Next Generation Project

The Active Badges system was originally invented and developed at the Olivetti Research Laboratory in Cambridge, UK, in 1990-92. It uses hardware infrastructure with the key components of infrared (IR) sensors installed in fixed positions within a building, and infrared emitters (*active badges*) worn by people or attached to equipment. The sensors are connected by a wired network that provides a communication path to the controlling device, called a poller. A poller can be implemented on a PC or a workstation using a software communication protocol to talk to the active sensors.

An active badge periodically transmits an infrared message containing a globally unique code (a badge identifier) using a data-link layer protocol running over an RS-232/422 network. The messages sent by the badges are received and queued by sensors. A poller periodically polls sensors and retrieves badge messages from the sensor queues. Each message and the identifier of the sensor that received the message is forwarded to the software part of the Active Badges system. The software layer maintains a database that maps sensors to places where sensors are installed, and badges to the users wearing them or the pieces of equipment to which they are attached.

The relation defined between a set of sensors and badge identifiers is called sighting. Using collected data, the system can infer where users or pieces of equipment are currently located. Information about the current locations of users and equipment is provided to various applications, such as presentation tools that display location data or applications that use location data to control the users' environment. The software part of the original Active Badge system developed at ORL uses the ANSAWare distributed environment.

The ABng project goal is the development of a new software layer of the Active Badge system that will be flexible and reconfigurable. To satisfy this requirement, ABng uses modern components and object-oriented technology. The system was developed in CORBA-compliant environments: Orbix, omniORB and OrbixWeb. It is based on the object model in which all logical and physical elements of the Active Badge system (users, locations, sensors, badges, etc.) are represented as CORBA objects.

<u>**Figure 1. An Active Badge**</u>

ABng Poller Implementation

In the context of the ABng project requirements, **ABng poller** must be implemented as a CORBA object. The poller provides two-way communication between sensors and the rest of the system. It is, in fact, a gateway between the ABS data-link layer protocol running over an RS-232/422 network and the IIOP implemented over a TCP/IP protocol stack.

The poller acts as a CORBA client when it forwards information from the sensors, and as a CORBA server during information flow in the opposite direction. Acting as a client, the poller provides information (mainly the badge identifiers seen by a given sensor) to all Main Sighting Processors (MSP) registered with the poller, so that the MSP can update the sighting relation. The observed/observer design pattern is exploited to organize this communication. As an example, let's see what happens when a person wearing a badge (see Figure 2) enters a room containing an IR sensor. The message containing the identifier, generated by the badge, is observed by the IR sensor and cached in its local buffer. When the poller sends the next periodic polling request to the sensor, the information cached by the sensor is sent back to the poller. The poller in turn sends this message, with the sensor and serial network numbers, to all its observers; the MSPs interested in receiving the information needed for the sighting relation modification.

Figure 2. Alex Laurentowski wearing Active Badge

Taking the role of a server, the poller provides operations that forward commands to sensors and badges. In the ABng system, these commands are issued by a dedicated server called Scheduler. The badge can play sounds, and turn one of its two LEDs on or off. This simple feature can be used to notify a person wearing an Active Badge about events—for instance, a phone call or an e-mail arrival. This notification could also include an action based on information about the person's location, e.g., a call could be redirected to the nearest telephone.

The poller server's software interface consists of two parts, each with its own functionality. The first part of the interface provides system operations that register MSPs interested in receiving information for the sighting relation update from the poller. It will also de-register them when they are no longer interested. The second part represents the operations corresponding to the set of commands performed by sensors and badges.

The ABng system is integrated using two services: a standard name server implemented according to COS OMG specification and a server manager. The server manager is a dedicated server providing a notification mechanism for all servers in the system interested in server activity. When a new ABng poller is started, it registers with the name server via the server manager; the server manager notifies all registered MSPs that a new poller has begun activity. The server manager also monitors the poller process using a simple keep-alive protocol. The poller's IOR (Inter-Orb Reference) is de-registered from the name server when the poller is removed from the system or when its process dies. This integration mechanism provides plug-and-play functionality for the ABng poller CORBA server, supporting self-configuration and automatic binding, which is crucial in systems with many dependencies between the objects.

Figure 3. ABng Architecture

ABng poller is a multi-threaded Linux application, with the main purpose of collecting sightings from the serial lines and distributing the received data to registered observers. This application can be configured to interface with sensors designed by the ORL as well as with the modified sensors commercially available from Olivetti. The six serial lines available in the prototype ABng poller can serve up to about 80 sensors with reasonable response time. When more sensors are needed, additional pollers must be deployed. The design of the ABng system places no limitation on the number of pollers; thus, system scalability is ensured.

ABng poller is able to cooperate with other parts of the system via the standard socket services or through the CORBA interface. This feature enables a smooth transition from the old ANSAWare-based location system to the CORBA-based ABng. Object Request Broker functionality is ensured by the omniORB which, although free, has performance levels hard to beat by the commercially available CORBA-compliant products including the industry leader—Orbix. An important feature of omniORB is its portability; it runs on most modern operating systems, including Linux.

Figure 4. ABng Poller

Hardware Platform

ABng Poller runs on the PC/104 hardware platform, which was developed in 1992 in response to a growing need for a more compact implementation of the PC bus, satisfying the reduced space and power constraints of embedded control applications. Yet these goals had to be realized without sacrificing full hardware and software compatibility with the popular PC bus standard.

The key differences between PC/104 and the regular PC bus are:

- Compact Form-Factor: card size is reduced to 3.6 by 3.8 inches. It has a unique self-stacking bus that eliminates the cost and bulk of backplanes and card cages.
- Pin-and-Socket connectors: rugged and reliable 64- and 40-contact male/female headers replace the standard PC's edge-card connectors.

- Lower bus drive current (4 mA): lowers power consumption to 1-2 watts per module.

In the ABng project, we use PC/104 modules manufactured by Advantech. Products manufactured by Advantech fall into several groups, from which we use two: Biscuit PCs and PC/104 modules. Biscuit PCs are a family of small, highly integrated single board computers designed for all sorts of embedded applications, and are equipped with the 80x86 processors—from 386 to 586. Since the ABng poller runs in a distributed environment, it was crucial to have an Ethernet controller on-board. Other components, such as a VGA adapter, printer port or floppy interface, are not necessary but are quite helpful in early development stages.

One of the first versions of ABng poller worked on a PCM-4822 Biscuit PC equipped with an AMD 486 processor, NE2000 compatible network adapter, enhanced IDE interface, two RS-232 ports, keyboard connector and 4-bit digital I/O interface. All these components, including 8MB of RAM, fit into a compact 145mm x 102mm board.

During the testing period, we found two main drawbacks to the PCM-4822 board. The board is equipped with a 120MHz 486 processor that requires an electric fan. As the only mechanical part of the ABng poller, the fan is the weakest part of the whole design. A second problem is the lack of a PC/104 interface on the PCM-4822 board. Adding any PC/104 modules requires an additional adapter. One of our main design goals was to create a robust device of minimal size, so we couldn't accept moving parts or unnecessary adapter boards. The solution came from Advantech, with the release of PCM-4823 boards, equipped with the PC/104 interface and an AMD 586 processor. The CPU has a heat sink attached, so an external fan isn't necessary.

To increase the number of sensors supported by the poller, we had to use a multi-port RS-232 module. The PCM-3640 manufactured by Advantech, with a price of about $30/port, seems to be a good choice. The only problem is that IRQ sharing is not implemented, so four unused interrupt lines have to be allocated for the module to work.

Linux Piccolo

The operating system platform for the ABng poller is Linux Piccolo, which has its roots in Red Hat 4.2. To fulfill the requirements of an embedded system, the size of Linux Piccolo has been reduced by keeping only the most important components such as system libraries, system commands, basic daemons such as telnetd, httpd, perld and necessary configuration files. The size of the first version of our system was about 25MB of hard-disk space. During the evolution of the system, its size was reduced to 4MB of compressed file-system image.

As Linux Piccolo runs on a diskless PC, two problems had to be solved: remote booting and root file system mounting. By remote booting, we mean downloading the Linux image from the boot server and performing a standard bootstrap procedure. After the kernel is loaded into memory, it mounts the root file system, which is crucial for the operation of the system. Without a local hard disk, the NFS root file system should be used.

The PCM-4822 board used in the first prototype of the ABng poller comes with the Remote Program Load (RPL) protocol burned into its boot PROM. The RPL protocol is supported by the following operating systems: Windows NT, Novell Netware, Microsoft LAN Manager and IBM LAN Server. Unfortunately, a Linux implementation of the RPL protocol does not exist. After an unsuccessful attempt to find an RPL specification, in order to implement it under Linux, we've decided to use Windows NT as a boot server. In our environment it wasn't a difficult decision, because we already had a Windows NT workstation. In other cases where an NT server isn't available, the alternative is to change the boot PROM image to support BOOTP/TFTP. A good place to find network booting solutions is the German company Imcon (http://www.imcon.de/). Their Boot PROM supports over sixty Ethernet cards, covering all major brands of current PCI and ISA cards, as well as many older 8-bit models.

After selecting boot protocol and configuring both client and server sides, two more elements are required: the Linux kernel and the root file system.

The kernel for the Piccolo workstation needs the following as a minimum set compiled in:

- NFS file system support
- “Root on NFS” enabled
- The Ethernet driver for the network card (For Advantech's Biscuit PCs, the NE2000 driver works fine.)
- RARP or BOOTP support for NFS-Root depending on your needs.

Additional parameters, such as the station IP address and the IP address of the NFS server, should be passed to the kernel at boot time. When **loadlin** is used to boot the kernel, the following command-line options should be used:

```
loadlin zimage nfsroot=/biscuits/piccolo1 \
nfsaddrs=149.156.97.54:149.156.97.58:: \
255.255.255.0:piccolo1:eth0:none
```

In this example, **/biscuits/piccolo1** is the path of the root file system on the server, 149.156.97.54 is the station address and 149.156.97.58 is the boot server address. The rest of the parameters are the network mask, the station name and the network interface name.

As the next step, the root file system of the diskless station has to be prepared, installed on the NFS server and exported to the diskless station. The Linux Piccolo file system can be downloaded from our web site (see Resources).

Another feature of the NFS root approach is the lack of a swap partition. With no swapping capability and only 8MB of memory installed, RAM is a very scarce resource; therefore, the number of processes running should be minimized. Apart from removing unnecessary applications, the number of **getty** processes has been reduced by editing the /etc/inittab file. After this thinning treatment, about 3MB of RAM was left to the ABng poller process.

DiskOnChip

The biggest disadvantage of the NFS root approach is that it requires both a boot server and an NFS server (which can be co-located on one physical machine) during both the boot process and the system operation. This approach is very convenient in the early stages of the development process, when home directories can be shared between a development system acting as an NFS server and the Linux Piccolo host. When the development cycle is over, a “snapshot” of the system, along with the required applications, should be taken and stored in the poller's non-volatile memory. The poller can then run independently of the NFS server.

The M-Systems' DiskOnChip2000 is a new generation of high-performance single-chip flash disks. The DiskOnChip MD2000 provides a flash disk in a standard 32-pin DIP package. Since the PCM-4823 board we use is equipped with a DiskOnChip socket, M-Systems' solution seems to be a natural choice. We soon discovered that there is one serious problem with DoC. DiskOnChip2000 has built-in TrueFFS (true flash file system) technology, which provides hard disk compatibility at both the sector and file level. It works in a variety of operating system environments such as DOS, Windows 95, Windows CE, Windows NT, pSOS+ and QNX. Unfortunately, Linux is not yet on the list of supported operating systems. The good news is M-Systems is planning to support Linux; by the time this article is printed, Linux should be supported.

As the deadline for our project neared, we couldn't wait for the support. The fact that DoC is not supported by Linux does not mean that Linux cannot be started from it. The solution was to create a DOS partition on the flash disk containing the Linux Piccolo kernel and the compressed file system image, and to use loadlin with the **initrd** capability. **initrd**, which stands for initial RAM disk, enables loading the RAM disk image by the boot loader. This RAM disk can be mounted as a root file system, from which applications can be executed. Using the following command:

```
loadlin zimage initrd=linpico.gz root=/dev/ram
```

the compressed kernel (**zimage**) and the compressed root file system image (linpico.gz) are loaded into memory. After those two elements are loaded, the kernel is uncompressed and executed. Code contained in the kernel is then used to uncompress and mount the root file system. When the file system is mounted, standard system initialization is performed.

Having experience with the NFS-root approach, we decided to buy a 10MB DiskOnChip. Relying only on the flash disk requires much more RAM than using an external NFS server. A 32MB RAM module has been installed on ABng poller. Also, the size of the whole Linux Piccolo file system should be reduced as much as possible to fit on the 10MB flash disk. All required files can be copied into the flash disk using an external floppy or hard drive, or they can be downloaded through the network.

To create a file system image, a block device is required. We use a loopback device for this purpose. The first step is to zero out the block device in order to achieve a better compression ratio. Then, the file system is created using the **mkfs** utility. The file system is then mounted at the temporary mount point and all required files are copied onto it. The last steps are to unmount the file system and compress it.

After each development cycle of the ABng poller application, the script in Listing 1 is invoked on the external Linux server to prepare the Linux Piccolo file system.

The biggest drawback of this solution is that no changes can be written from within Linux. In the case of ABng poller, this is not a big problem, but for other kinds of applications it might be unacceptable. In such a situation, flash disks with an IDE interface may be used. An example of such a flash disk that performs well under Linux is the SanDisk FlashDrive.

Conclusions

Linux wasn't chosen just because it is free and very popular in the academic community. Linux was the best choice for several reasons. First, we needed a multi-threaded operating system that could be tailored to our own requirements. With Linux source code available, this customization could easily be performed. The other important features of Linux are its mature networking subsystem and the availability of the CORBA environment. The performance of ABng poller turned out to be better than expected even with a limited amount of RAM. The customization and integration of different components of the system were performed effectively without any major technical problems. This proves Linux to be an advanced and open operating system.

Acknowledgments

This work is supported by Olivetti-Oracle Research Laboratory, Cambridge, UK.

Igor Bokun lives in Krakow, Poland, where he studied computer science at the University of Mining and Metallurgy. He is currently working on a Ph.D., focusing on networking and multimedia. He has been using Linux since 1991. In his spare time, he enjoys windsurfing and scuba diving. Igor can be contacted at bokun@ics.agh.edu.pl.

Krzysztof Zielinski has been a professor at the Institute of Computer Science, University of Mining and Metallurgy in Krakow, Poland since 1992. His research interests focus on distributed computing systems and networking, including object-oriented distributed systems and multimedia applications. His current research interests concern CORBA-compliant systems and services. He can be reached at kz@ics.agh.edu.pl.

The Future of Linux

Greg Roelofs

Issue #54, October 1998

An informal report on the panel discussion held in Santa Clara on July 14.

The Future of Linux was set up as a panel discussion and was held at the Santa Clara Convention Center (in the heart of Silicon Valley) on the evening of July 14, 1998. It was hosted by Taos Mountain and the Silicon Valley Linux Users Group (SVLUG), and was sponsored by them, Intel, Red Hat, *Linux Journal* and VA Research. Apparently it was considerably more popular than Taos expected; people stood in line for up to an hour to register, and the free food and free VA Research/Linux T-shirts ran out. I didn't get a firm count, but Taos said 850 people had returned their RSVP, and my estimate is 700 to 900 attended.

Linus Torvalds and the Panel

The panel was a distinguished group: Jeremy Allison, one of the lead Samba developers; Larry Augustin, founder of VA Research and member of the Linux International Board of Directors; Robert Hart, from Red Hat Software; Sunil Saxena, from Intel's UNIX Performance Lab; and, of course, The Man himself, Linus Torvalds. (While I know there are a lot of Linux fans who like to pronounce “Linux” with a long “i” sound (LYE-nucks), and despite the fact that

Linus himself doesn't care how anyone else pronounces it, he unquestionably did so with a short “i” as in “linen” (LINN-ucks). In Swedish he presumably still pronounces it the third way, roughly “LEE-nooks.”) The discussion was moderated by Michael Masterson of Taos, who traded off questioning duties with Phil Hughes, all-around hairy guy and publisher of *Linux Journal*.

Linus Torvalds responds to a question from the audience. Ben Spade, president of the Silicon Valley Linux User Group, serves as microphone wrangler.

I did not have a tape recorder, so answers are paraphrases of what was actually said. [*Comments in square brackets are my personal asides.*]

Phil Hughes' Questions

1. How much bigger will the Linux market be in 2000?

- Jeremy Allison said 20% to 25% of shipping Intel systems will have Linux pre-installed.
- Linus Torvalds said that he's always been bad at predicting things and basically weaseled out of answering the question.
- Sunil Saxena also declined to speculate.
- Larry Augustin said that Linux would be the #1 UNIX by 2000.
- Robert Hart mentioned the Datapro report that showed only two operating systems increased their corporate market share in 1997; Linux was one of them. He said the doubling time was 12 months, which would imply between 20 million and 40 million Linux users; “I'll be surprised if we don't go beyond that.”

2. World domination: how much longer? [in reference to Linus' rather famous stated goal in his .sig or .plan or something]

Linus Torvalds: “That used to be a joke... [*much laughter*] ...and it's becoming less and less so.” He said his ego hopes it will happen in five to ten years; but more realistically, he hopes that in five to ten years no one dominates the industry.

3. What is Samba's role in Linux's acceptance?

Jeremy Allison first asked for a show of hands; it appeared that roughly 40% of the audience used Samba. Then he gave the short answer to the question: Samba “essentially allows people to remove NT servers.” He noted that SGI is officially adopting Samba [*recall that they, like HP, are now selling NT systems as the low end of their product line*] and that “some crazy folks are running it straight off of CDs with 200 users” (mostly universities who “do not want NT”). By the end of the year, he hopes that Samba will be able to completely replace all primary NT Server functions.

4. Open Source is obviously just another fad—isn't it?

Larry Augustin was the first to disagree; he said that Open Source is here to stay—for example, it allows a company like Netscape to compete on its own terms with Microsoft, not on Microsoft's terms. It also supports a Darwinian model: if one vendor's support is lacking, you've got the source and can take your money (and business) elsewhere. That's not possible with the closed-source model epitomized by Microsoft.

Robert Hart expanded on that point: it's all about control. If you need a new feature or bug fix or other customization, you can simply hire someone to do it for you. “You don't need anyone's permission; just do it!” I believe he related an example of a company with a large application that was in dire need of a bug fix; they were willing to spend essentially any amount of money or manpower to get the thing working, but their vendor was unresponsive and they had no real alternatives.

Jeremy Allison claimed that he was fundamentally “a lazy programmer” and that the Open Source model is a way of letting users do the work [*more laughter*]. He mentioned that some incredible Samba patches occasionally turn up in his e-mail—often oddball customizations useful to only a few people, but to them they're extremely useful. “Imagine asking Microsoft to do a custom NT Server for your site.”

5. What will be the long-term effects on Linux of Microsoft's recent win against Netscape (i.e., bundling MSIE in Win98)?

Linus Torvalds dismissed the Department of Justice and the U.S. legal system as important factors in Linux's future; “the only thing that will matter is the market.” In fact, he claimed it's an advantage since there are many companies who find it hard to compete, when Microsoft sees what they're doing and simply incorporates similar technology directly into the OS. In the Linux arena they can find a niche and compete (echoing Larry's comments above), as Corel has, for example. “That's one reason why, in the end, a monopoly just doesn't work. [*pause*] Maybe that's just me...”

6. What do we need to do to get applications (such as from Adobe and Quark, which are the only non-Linux applications used by *Linux Journal*), ported to Linux?

Robert Hart said there are just two things: let them know you want Linux ports, and show them there's profit to be made.

Larry Augustin related an article seen on Slashdot earlier in the day about Informix's unannounced Linux port and said the key is to tell vendors, “If you port it to Linux, we will buy it.” [*Three days later, Slashdot and InfoWorld Electric reported a sudden reversal of plans at Oracle: they will be porting Oracle 8 to Linux after all. In fact, they say they've had it running internally for a while already. See also InfoWorld Electric's article on Informix's official Linux announcement, made on July 22.*]

7. With regard to the Linux Standard Base (a standard for base-level compatibility across Linux distributions): Red Hat and Debian's standard

package formats, Red Hat's early adoption of glibc vs. everyone else, etc., are we doing this right? Are there too many Linux “standards”?

Robert Hart had three points in response. First, a lot of discussion goes on between the various distribution makers, precisely for the purpose of avoiding fragmentation. Second, we have a danger of stultifying and crushing the rapid pace of development and the incredible customization choices available to users if we have too much rigidity and standardization. Third, to the other distributions: “Please get with it—glibc is the only actively maintained C library.”

Larry Augustin countered that he's seen a lot of users who, when they upgraded to Red Hat 5.x, found that “everything broke”. [*Thanks to Jason Riedy for the reminder that just installing the older libc 5.4.x somewhere in the library path isn't sufficient; most shared libraries used by older applications need to be duplicated, as was the case in the changeover from a.out to ELF binaries a couple of years ago.*] “You're in the big time now. Some things (like Informix) users can't simply recompile—try to make things easier for people and remain compatible.”

8. What if Microsoft plays the Linux game? For example, Open Windows 99 or Internet Explorer for Linux?

Linus Torvalds first noted that he's working at a company [*Transmeta*] whose product won't be available on the Internet. He went on to say that he has a lot of respect for Microsoft's PR machine, and “let's hope they do.”

Jeremy Allison apparently interpreted “Open Windows 99” as a hypothetical Microsoft release based on Linux, and said he would welcome MS Linux—the GNU General Public License (GPL) limits abuse. “If they change it, we'd get the source code,” to which Linus muttered, “We could fix it, too.” [*much applause and laughter*]

9. NASA, NIST, the U.S. Postal Service, the IRS all use Linux—is the U.S. Government the first step toward world domination?

Linus Torvalds: “I hadn't really thought of that, but now that you've planted the idea...” [*more chuckles*]

Prepared Question #1

Is Linux superior, comparable, or inferior to commercial operating systems? [or something like that]

Jeremy Allison (Samba): [*I think he was one of the panelists who made the comment that "Linux is a commercial OS"; his answer amounted to choice #1:*]

Linux is very standards-compliant (e.g., Posix); a good approach is to develop first on Linux, then port to a proprietary UNIX system. For example, Samba has three separate pieces of code to deal with some aspect of file-system stuff; Linux supports all three interfaces, so they just choose the one that runs the fastest.

He gave a wish list of improvements he'd like to see, though: 64-bit file-system support ("for those 20GB Exchange databases"); access control list (ACL) support; asynchronous I/O support; NFS file-locking and improved performance [*amen to that*]; and a thread model like Solaris.

He noted that Linux has current support for more platforms than any other OS: x86, SPARC, Alpha, Power PC, 68k, etc. Bugs are fixed the fastest, especially security leaks.

There's been a Linux "iBCS" module to support SCO UNIX binaries for a couple of years; at this year's USENIX, SCO announced (and demoed) a module to run Linux applications.

Larry Augustin (VA Research): His answer was "yes". Linux is not (yet) as far along as Solaris in supporting 64-way symmetric multi-processing (SMP). [*I thought the SPARC-based Fujitsu AP1000+ on which David S. Miller reported success last year was a big SMP box, but as Jason Riedy pointed out, it's a distributed-memory multi-computer similar to the Connection Machine CM5.*]

In his slides of user ratings (Datapro survey, which was mentioned several times during the evening), Linux was not only the overall winner in a field of half a dozen operating systems (Windows NT placed last), it also won in all but two categories—and only Digital UNIX was rated higher in those two (availability and performance). The other categories included reliability, technical support, price, etc.

Robert Hart (Red Hat): Linux is a commercial operating system. It is sold and supported commercially (Red Hat, Caldera and others); it is used commercially; its only difference is that the source code is freely available.

Key "commercial OS" features like a journaling file system and database access that bypasses the file system layer are coming very soon.

Many of Red Hat's users (more with every release) have never installed an OS, which means Red Hat has to "reverse engineer" their hardware configuration.

Why is Linux not ubiquitous? It's still not suitable for everyone (he mentioned his “75-year-old mum”), and although there are good office applications for Linux, there aren't any killer ones yet. (There were follow-ups to the suitability comment by some of the other panelists.)

Sunil Saxena (Intel): He presented some slides that amounted to a “yes” response as well.

Strengths: Linux is becoming the OS of choice of ISPs; on 32-bit Intel systems, Linux has broader device-driver support than any other UNIX (e.g., SCO, Solaris/ x86, etc.); its Open Source model means that updates, patches and bug fixes happen in “Internet time.”

Weaknesses: SMP support and scalability is still evolving (although he noted that Leonard Zubkoff did a successful two-day port to the brand-new, four-way Pentium II Xeon system that Intel and VA Research showed off); good server management is missing (e.g., using a remote serial line or modem to update things, including the BIOS); drivers for high-end hardware tend to be lacking; and large-memory support (say, multi-gigabyte range) isn't there.

Making it better: he said (and repeated several times throughout the evening) that Intel really wants to help and do more to support Linux, and in particular, they see the following as likely areas of collaboration:

- more than 4-way SMP (serious scalability, at least 16 to 32 processors)
- drivers for high-end platforms
- direct server control and management
- support for PII features such as 36-bit addressing (up to 64 GB of RAM), enhanced system calls and save/restore, MMX instructions, the page attribute table, and on-chip performance monitors

Linus Torvalds: He started off with a comment to the effect of, “What can I say? I came to listen to the others.”

He noted that Linux originally was a one-person OS; it was never intended to be useful to others. He also pointed out that it has just shown up on the list of the world's most powerful supercomputers (in a cluster design); he thought it made #316. [*Actually #315 in the June 1998 list—see the press release for details.*] At the other end of the spectrum, it's being ported to Palm Pilots. “I don't see that [*its broad portability and usefulness*] ending any time soon.”

He responded to a couple of Sunil's comments:

- 36-bit addressing on Intel: “We've been doing that on Alpha for awhile.”

- Page Attribute Table: he didn't know about Intel's implementation but said that he'd suggested it to an Intel engineer a few years ago; “I don't know if they did it right, but if so, I'll be happy to use it.”

Prepared Question #2

Where will Linux be in two to three years?

Sunil Saxena: “It's all on my foil right here.” Among Internets/Intranets/ISPs, Linux will continue to grow, especially in corporate America; we'll see continued adoption.

There will be new growth in the areas of E-commerce and business-to-business Internet EDI (electronic data interchange).

It will start showing up in all sorts of Internet appliances, including wearable computers, video-conferencing systems, etc.

We'll see 64-bit Linux on the IA-64 (Merced).

Linux will move into the data center via high-availability clusters and 16- to 32-way SMP systems.

Other spiffy features like I2O, hot swap, serial-based server management and control, etc., will be supported.

Linux developers will be granted early access and increased access to specs and Intel engineers.

“Please come talk to us and tell us what we can do.”

Larry Augustin: One big prediction: kernel 2.2 will be released within three years (“and that's pushing it”). [*much laughter*]

He also took the opportunity to thank Leonard Zubkoff for the four-way Xeon port being demoed by VA Research in the rear. Apparently he modified (added?) 20 lines of code at the last minute without even having all of the source there.

Robert Hart: Two to three years is an eternity for Linux—even a week is a long time.

People are lazy, and laziness leads to creativity, which means great improvements for Linux.

He again noted that it's the only non-MS operating system to gain market share and that it's being actively courted by large vendors such as Intel. He also noted that it's generating strong media interest, and not only that, but the reporting is generally both accurate and useful. (He thanked the representatives of the press in the audience.)

All photographs used with this article are by Jim Howell, courtesy of Taos Mountain.

There will be a strong showing of easy-to-use applications.

Linux will be the dominant server platform, not just on Intel but across all platforms. I believe he mentioned “64-bit, 4-way Merced,” too.

He mentioned that it was Bastille Day and recalled how, in 1788 and 1789, the people rose up and stormed the IT department and gave freedom to oppressed machines. “They even executed some people.” But he disputed his own comparison and said that Linux isn't so much a revolution as an evolution; it didn't happen just once.

He concluded by predicting that in two to three years, Linux will be “very nearly everywhere”.

Linus Torvalds: “I'm really bad at predictions.” For example, a few years ago when he was asked about SMP support, his reaction was, “I don't know, it's too expensive”; he said he didn't care much about it and didn't find it interesting. But for the last year he's worked almost exclusively on SMP.

2.2 will be out by then. [*much more laughter*]

The kernel is really just a vessel for what one can do; he claimed that it was inappropriate to ask him such a question—“what's really exciting is applications” (and the journaling file system is “slightly exciting”).

Servers will be big. [Do we sense a theme here?]

The interesting part will be “pretty” applications that traditionally haven't been UNIX-based.

Jeremy Allison: He used to keep track of every minor release of every piece of free software; these days, no one can do so for even 10% of it, and there will be some incredibly cool things in that other 90%.

Linux will be a “killer server platform.”

Everyone at Cisco Systems uses Linux every time they print, whether they know it or not. [*See the c.o.l.a announcement for details about the printer-administration tool used at Cisco, and the article about Cisco in this issue*.]

At least one major PC vendor will start shipping PCs with Linux pre-installed. (If not, Robert will start such a company.)

With regard to applications, vendors are already “not as greedy” as they've traditionally been on UNIX systems; Linux applications are priced similarly to Wintel applications, not five times more. Linux will be on the desktop, and one will be able to buy almost any application for Linux.

Again, he predicted that Linux will be installed on 20% to 25% of shipping Intel systems.

Resources

Greg Roelofs escaped from the University of Chicago with a degree in astrophysics and fled screaming to Silicon Valley, where he now does outrageously cool graphics, 3D and compression stuff for Philips Research. He is a member of Info-ZIP and the PNG group. He can be reached by e-mail at newt@pobox.com or on the web at pobox.com/~newt/.

Linux Print System at Cisco Systems, Inc.

Damian Ivereigh

Issue #54, October 1998

Cisco runs a redundant system of 50 print servers using Linux, Samba and Netatalk. It prints to approximately 1,600 printers worldwide, serving 10,000 UNIX and Windows 95 users, some of whom are in mission-critical environments.

At your office, can you print to the nearest printer, or do you have to use a printer set up just for your desktop machine? If you wanted to send a job to a printer in another department or in another office, could you do it as easily as with your local printer? In most large companies, the answer would be no.

System Administrators seldom want to take care of printing problems, and there is rarely anyone else to turn to for help. Out of desperation, you're determined to fix a problem yourself, only to be told, “My colleague tried that, and broke printing for a week. Don't touch it.” In short, printing has become a Pandora's box no one wants to open. It is surrounded by more folklore and black magic than any other area in modern computing.

In this article, I will describe the general problems with printing in a large corporate environment and the general methods for solving these problems. I will then detail how I solved the particular problems at Cisco. Using software almost entirely downloadable from the Internet, I produced a highly visible, mission-critical, fault-tolerant print system used every day by over 10,000 people worldwide.

In a nutshell, the solution depends on multiple Linux servers which, by communicating with each other, effectively work as a single “distributed machine”. This approach offers one solution to many problems inherent not only in print systems but, more generally, any network resource (e.g., mail, disk space, etc.) in a large corporate network.

A distributed machine may sound complex, but very little magic is involved. As we shall see, the “magic” comes from applying the traditional UNIX method of combining many little pieces into a whole significantly greater than its parts.

Printing Strategies

First, let's talk about some of the general problems that people face when printing in a corporate environment. Printers are based on mechanical parts that are slower and less reliable than the computers sending jobs to them; thus, queues and frequent status updates are required.

Although vendors are trying to create standards (for example, the work of the Printer Working Group [PWG]), the current standardization is poor, and printer manufacturers are programming their machines to talk to as many different standards (or emulations) as possible. Few of these emulations work well or allow good user control of the printer.

Essentially, there are two main strategies for organizing printing: direct client-to-printer or via a central print server.

1. Direct Client-to-Printer

The client connects directly to a printer and takes complete control as it sends its print job. Any printer status is sent directly back to the client. Once the printer has finished, the client disconnects and the printer is then available for another client.

1a. Advantages

This method is simpler for use in small offices. Each person's machine is set up in isolation of everyone else's—no need to think about any larger issues. Since each user is isolated, problems usually affect only a single user. Provided his printer does not fail, an individual user can carry on printing, regardless of problems others may be having.

Each user controls his own queue directly. He can cancel a job on his own machine using whatever tools the operating system provides.

1b. Disadvantages

With a direct client-to-printer system, performing any global changes is difficult. If the IP address of the printer changes, the engineers have many client machines to track down and reconfigure.

Each client machine has to compete for the printer. If the printer is already busy when a client tries to print, the client has to keep retrying until the printer is free.

Providing an orderly queue is difficult. Since all clients get control of the printer in a random order, there is no guarantee of when any particular client's job will start printing, or that jobs will be printed in any particular order. It is almost impossible for a client to know that other clients are waiting for the printer.

A client can print only to the printers that support its protocol. For example, an Apple Macintosh can talk only to the printers that support AppleTalk.

Tracking down and canceling unknown jobs is also not easy. For example, if a printer is busy printing a 2,000-page document, the sending client is not apparent, and to actually stop the print job, the system administrator has to get appropriate permissions on that client machine.

A fix to one protocol can break others. Service technicians, who tend to be experts in their own field, can easily break things for other protocols. For example, an engineer may reset a printer to fix a Novell printing problem, and in the process break the TCP/IP setup (which he doesn't understand anyway).

Certain printers cannot switch well between different protocols. Some printers, particularly the older ones, have even been known to crash completely (requiring a reboot) when switching between TCP/IP and AppleTalk.

2. Central print server

The client sends its print job to the central print server and disconnects. The print server takes the job and adds it to the queue for the designated printer. The print server then connects to the printer, and sends the job. Any status is sent to the print server, not the client.

2a. Advantages

Since the print server has significant storage capacity, it can receive jobs at any time, regardless of what the printer is doing. The client machine can send the job, then move on to another job.

The jobs go through a central queue, which prints them in the order received. Each user should be able (operating system permitting) to see all jobs waiting to print on a printer by looking at this print server queue.

A system administrator may kill any job on the print server, regardless of its source.

If a printer fails, it is easy to re-route all the jobs from the broken printer to a working one.

Any printer changes can be made on the central print server alone, since this is the only machine that talks directly to the printer.

2b. Disadvantages

A central print server system is more complex. It requires a system administrator to set up the print server and keep it running.

If the print server dies, all printing stops, unless a good backup print server is available.

The users have no queue control. Menial tasks such as print job cancellations fall on the shoulders of system administrators, if the users no longer have the permissions or skills to do it themselves as they do in the direct client-to-printer case.

Usual Print System

Most larger companies make a half-hearted attempt at the central server approach. The real problems begin when more than one “central” server is implemented. The UNIX system administrator sets up a UNIX print server, the Windows guy sets up an NT server, and some of the clients skip the servers completely and go directly to the printer. All jobs meet at one printer, where chaos ensues.

You now have all the problems of the central server approach compounded with all the problems of the client-to-printer approach plus a few extra thrown in for good measure. Printer changes must be implemented on multiple servers by multiple system administrators leading to multiple potential errors. Multiple machines (now servers instead of clients) compete for the same printer, there's no orderly queueing and we still don't know where that 2,000-page document is coming from.

To make matters worse, each environment has a different name for the same printer, which makes tracking down printers even more difficult. When a user has a problem, he most likely doesn't know which environment he is trying to print from. He'll call the wrong system administrator, who can't find the user's printer name in his environment. The system administrator will suggest the user call a different group, who will pass the user to another group, and so on. Five system administrators later, the user is back to the first one. Overall, a frustrating experience for everyone. This situation was beginning to occur at Cisco.

Ideal Print System

After a few months of dealing with these problems, I decided to find a better way. I sat down and detailed what I believed to be the “ideal print system”. It had to have the advantages of the server approach, yet mitigate some of the disadvantages.

- **Multi-protocol**: The server must talk to all the different protocols available to both clients for sending and printers for receiving.
- **Ultra-reliable**: Use redundancy to remove the single point of failure inherent in most central server approaches.
- **Single point of queueing**: No matter where the job comes from or the route it takes, all jobs for a particular printer must land in a single queue handled by one machine.
- **Expandable and flexible**: Cisco is a growing company. Any system has to be able to scale well and allow frequent reorganization.
- **Centrally, de-centrally and remotely manageable**: Cisco has offices worldwide, some of which have local expertise, some of which don't.
- **Cheap**: The system has to be affordable for the small offices, yet expandable for use at headquarters.
- **Queue management devolved to the users**: System administrators don't have time; users want control.
- **Avoid duplication**: Any information duplicated by hand is prone to error. Even entering the IP address into both the printer and the print server should be considered a duplication.
- **Simple to manage**: No matter how many servers are added for redundancy or capacity, the management of these must remain simple.

Initial Cisco Configuration

When I started at Cisco, the printing was becoming difficult to manage (see Figure 1). While Cisco was no worse in this respect than most companies, they depended on printing for their manufacturing process. They knew it was very important for the system to work efficiently. Thus, I was taken on to maintain and improve on the existing UNIX printing, which used two small SunOS-based print servers.

Although these two servers by no means controlled the entire printing at Cisco, since I was a dedicated print system administrator, it was widely assumed that I did.

Figure 1. Original Printing Configuration at Cisco

The company's big UNIX servers were sometimes printing to one print server, sometimes to the other. The UNIX workstations were printing either via the print servers or directly to the printer. The Apple Macintoshes (used extensively on the desktop) always printed directly. Most of the printers recognized only AppleTalk, so Gatorboxes (from Caiman Systems) were used to translate UNIX print jobs into AppleTalk.

Caiman Systems had gone out of business and the Gatorboxes were intermittently crashing. My predecessors had started to enable TCP/IP protocols on some of the Hewlett-Packard (HP) printers so that the UNIX print servers could talk to them directly. Doing this required either plugging an IP address into each printer (via the front panel) or setting up a **bootptab** entry in each print server so that a printer could find its IP address using the BOOTP protocol (see Glossary).

In theory, one print server was the main one and the other was a backup. However, these two servers were substantially different in configuration. Duplication of the setups was manual, i.e., one had to configure the print queue on both machines. Some central UNIX servers were queueing to the "primary" and others to the "backup" print server. A few printers were set up for printing on some of the central UNIX servers and others were not. I spent much of my time tracking down print problems, only to find they usually came down to an incorrect configuration.

Cisco had never directly instructed me to design a new print system. They just asked me to make sure printing worked. They trusted me to do whatever I felt necessary. My motivation for improving it was simply that I find repetitive tasks boring and unfulfilling. I find nothing more frustrating than treating the symptom, while ignoring the disease. I never decided to throw out the old system entirely, I just slowly improved on it—tackling the biggest problem of the moment.

Remove Duplication in the Client System

Each printer needed to be individually set up on each UNIX server (the LPR client). This meant a lot of manual work, either when setting up a new UNIX server or creating a new printer. I looked at the client LPR system and realized it had a very simple function: just forward the print job to the print server.

Here is a typical /etc/printcap entry for the printer "foo", which sends the job straight on to the print server "prntsrv":

```
foo:\
    :mx#0:\
    :sh:\
    :sd=/var/spool/lpd/foo:\
    :lf=/var/spool/lpd/foo/log:\
    :lp=/var/spool/lpd/foo/.null:\
    :rm=prntsrv:\
    :rp=foo:
```

The only item which changed when using a different printer is the word **foo**.

I took the LPR source and replaced the routines that look for the entry for a particular printer in /etc/printcap with routines that faked the entry. If LPR asked for the printer “bar”, my routines would return a printcap entry much like the one above, but with **bar** in place of **foo**. The only other variable was the name of the print server which was looked up in a master configuration file I created for the whole system. There were a few other things to do, such as creating a spool directory, but essentially this is all the work the routines did.

The remainder of the LPR code proceeded as before, not realizing anything had changed. Since I hadn't touched the remainder of the code, I had very few bugs. I had removed a large source of information duplication, and I could now be sure that all the company's printers were available on all the central UNIX servers, with print jobs being sent to the correct print server.

Note that the client will also accept a print job for a non-existent printer (it doesn't know the difference) and send it to the print server. The print server will reject the job, but will not say why (the protocol doesn't allow it). The client keeps retrying for 48 hours before finally rejecting the job and e-mailing the user. This is not an ideal situation but was acceptable at Cisco.

Remove Duplication in the Print Servers

I next tackled the duplication within each print server. Several files were used to control the printers: /etc/printcap and /etc/bootptab as well as some setup files used by each printer. Each file contained the same information in a different format.

There were three different ways of talking to the various printers as set up in the /etc/printcap file:

- JetDirect for HP printers using their TCP/IP JetDirect interface
- Raw TCP for serial printers attached to TCP/IP-to-serial converters
- LPR protocol for EtherTalk printers (attached to Gatorboxes) or to remote print servers not under my control

To eliminate duplication, I needed a master configuration file from which I could generate the various configuration files required for the three protocols. The master configuration file contained all information required by any protocol, such as the name, IP address, Ethernet hardware address, the remote server and remote printer name. I created a script, **mkprint**, which generated all the other configuration files, created the spool directories and more.

Not only was this method simpler than editing the individual files manually, it was much less prone to error. For example, since the IP address supplied to the LPR system was the same one supplied to the BOOTP system, they had to match. I could not get them wrong.

Netatalk

As I mentioned earlier, the Gatorboxes were causing many problems. Often, an individual print queue on the Gatorbox would stop receiving print jobs and require someone to log on to the Gatorbox and restart it. They had significant “memory leak” problems which would cause them to run out of memory and crash, requiring someone to physically go to the box and press the reset button.

I researched first the Columbia AppleTalk Protocol (CAP) and then Netatalk as possible ways of getting the print servers to use EtherTalk directly. Both seemed to work well, but Netatalk was being more actively developed and required less load on the machine through the use of kernel-level drivers.

By installing Netatalk and modifying the mkprint system to allow for this new type of printer, I could remove the Gatorboxes from the loop. Another duplication had been removed.

PCs and the Web

Cisco started introducing desktop PCs, and although I did not realize it at the time, this was the beginning of a major push to change from Macintoshes to PCs. The Desktop Technology Group, who managed the PCs, introduced an NT server as the PC print server and I started getting calls to kill print jobs that did not exist on my servers.

I forged some links with the Desktop Technology Group and suggested they redirect the print jobs from their NT server through my print server, rather than going directly to the printers. I could now see and cancel all the print jobs. However, the NT server did a poor implementation of the LPR protocol. For one thing, once it had sent the print job, it considered that job printed, and the users could not see from their PCs either their own job or the total print queue.

To mitigate this loss of visual feedback, I created a simple web page which asks for a printer name and then displays the print queue by using the output of the lpq command. All other information for the page was generated by extending mkprint again.

Figure 2. Ideal Print System

Now I had a much tidier print system, with all jobs following a well-defined path from client to printer. The ideal print system had taken shape (see Figure 2).

Location Codes

Cisco decided to expand its facility in Research Triangle Park (RTP), North Carolina to include a full data-processing center. It seemed sensible to place a print server there to reduce print traffic across the WAN link.

I created a complete duplicate of one of the print servers I had in San Jose for this RTP print server. Recognizing the impending manageability problems with multiple servers and the requirement for quick backup in case of server failure (failover), I decided to organize printers into more manageable groups based on physical location. I called these groups “location codes” or “loccodes”. I assigned each loccode group of printers to a single server, which actually sends jobs to those printers. Any other servers receiving jobs for those printers would simply forward the jobs to that designated server. The advantage of this system was that, in case of a server failure, I could now move these loccodes (and their associated printers) quickly to another operational server, providing failover capability.

The steps I took to implement this system were quite straightforward. I first copied the master configuration files over to each print server using **rcp**. I then modified mkprint so that it took note of the loccode and its associated server as it created the /etc/printcap entries. For each printer, it extracts its loccode and figures out its assigned server. If that server is the one mkprint is running on, mkprint proceeds to create a printcap entry as before; otherwise, it creates an entry which simply forwards the print job to the appropriate server.

For example, suppose the printer **foo** had a loccode of **SJK2** (San Jose, Building K, 2nd floor), and the loccode is assigned to server “print-sj” in the list. The printcap entry for this printer on the server “print-sj” would be as before:

```
foo:\
    :mx#0:\
    :sh:\
    :sd=/var/spool/lpd/foo:\
    :lf=/var/spool/lpd/foo/log:\
    :lp=/var/spool/lpd/foo/.null:\
    :if=/usr/local/atalk/ifpap:\
    :of=/usr/local/atalk/ofpap:
```

However, the entry on any other print server would be:

```
foo:\
    :mx#0:\
    :sh:\
    :sd=/var/spool/lpd/foo:\
    :lf=/var/spool/lpd/foo/log:\
    :lp=/var/spool/lpd/foo/.null:\
    :rm=print-sj:\
    :rp=foo:
```

The last two lines tell the **lpd** program to forward the job to the print-sj print server.

Thus, it did not matter which print server a job landed on; it would automatically be forwarded to the “correct” print server. Using this scheme, there was no way a printer could receive jobs from more than one print server. This immediately provided the single point of queueing mentioned previously as part of the ideal print system.

I wrote a simple script called **allmkprint** that would copy the master configuration files and run mkprint using **rsh** on all print servers. I extended the web interface so it too would realize if it was being asked for a printer residing on a different print server, and forward the user's browser to that print server.

Now, when a server died, I just moved all the loccodes across to another print server and ran allmkprint—a simple failover system was in place.

Samba

Every time a new printer was installed, not only did I have to create a queue on my UNIX print servers, but the Desktop Technology Group also had to create a separate queue on their NT servers. To make matters worse, the drivers on the PC had to be carefully matched with the drivers used under NT.

I investigated Samba and was extremely impressed with its ability to provide the same services to PCs using the same protocols NT used. Essentially, one could make a UNIX machine pretend to be an NT server. Samba is also extremely configurable—so much that it is easy to be confused by the enormous array of choices.

Expecting rejection and a passionate argument of NT versus UNIX, I approached the Desktop Technology Group with the idea of taking over PC printing. Their response was, “Really? You'll take over setting up and managing these obnoxious printers for us? Hey, it's all yours!”

The Samba protocol (SMB), however, has a severe limitation: the browsing (which allows the user to get a list of available printers) is done using a single UDP packet, and so is limited to about 8KB worth of printer names and descriptions. In our environment, this translated to about 50 printer names per server. However, I found Samba had the ability to use different configuration files according to the name the PC thinks the server is called. For example, suppose that for one physical server, I register the two pseudo-server names pserver1 and pserver2 in Microsoft's version of DNS, WINS. The PC will see two servers, pserver1 and pserver2, both referring to the same physical Samba server. The Samba server will pick a different configuration file and serve a different set of printers according to which pseudo-server the PC thinks it is talking to. This allows us to effectively break the 8KB barrier by separating printers into smaller groups.

The printing itself is simple. The Samba configuration files specify a program or script to run when a print job is received. Usually, this is a simple **lpr** command. The PC queue display is done by converting the output of the lpq command.

The print servers could, of course, talk to all the different printer protocols. Add to this the Samba capability of receiving jobs from any PC, and now any PC can send a job to any printer—even an AppleTalk printer.

Service Groups

I could have just associated the pseudo-server names with the loccodes; however, I found this lacked the flexibility I needed. Since the PC printer path (in the form of "\pseudo-server\printer-name") was fixed in people's PCs, I found that if I moved a printer from one loccode to another, people would have to re-install the printer on their PCs. This caused a problem when I was making administrative changes like splitting a loccode up or joining two together; in this case, I wanted to be able to make these changes without involving the user.

The only way to do this was to associate the pseudo-server name not with the loccode, but directly to another grouping: the "service group" or "sgroup". I could associate a loccode to one or more of these sgroups. Conversely, each sgroup could have multiple loccodes associated with it. To use database parlance, loccodes and sgroups have a many-to-many relationship. The sgroup concept allows me to split, join or move loccodes without changing the client PCs.

For more information on sgroups and loccodes as well as an example of how they are used, see the "Loccodes and Sgroups" sidebar.

Enter Linux

Samba, and all these damn sgroups, were beginning to put a sizable load on the print servers. We had upgraded the print servers at our headquarters in San Jose to two Sun SPARC 20s; however, the growth in PC usage was soon stressing even these two. To make matters worse, Windows 95 has a very short time-out period when asking for a listing of the printer queue (less than three seconds). If it times out, the printer is marked on the PC as "off-line", and the user is required to go into the settings and put the printer back on-line. This feature was generating too many calls.

I now introduced a couple of Red Hat Linux machines as Samba print servers. The servers were HP XM4s: 120MHz Pentiums, with 32MB RAM and 1GB hard disk. These were the same PCs that were used as the standard desktop machines at Cisco.

I had not ported the entire printing system from SunOS to Linux, so I could not use them as final print servers—that task was left to one of the two SPARC 20s. The Linux print servers would receive the print jobs from the PCs using Samba and then forward them to the SPARC print servers. This was possible due to the sgroup and loccode system I explained earlier.

Simple Distributed Database (SDDB)

Due to the growth of Cisco and the sheer amount of day-to-day print administration, I had allowed the engineers access to edit the master configuration file. However, this was creating a problem. Since the file was edited with **vi**, there was no locking, and I was beginning to have problems with people overwriting each other's changes. Also, the file was getting so big the mkprint program was taking a significant amount of time to run. I needed to put this configuration data into a database.

I wrote what I thought would be a “simple distributed database” (SDDB). I soon discovered the first two words are a contradiction in terms. It is actually more like a “network directory” than a database—it performs a function similar to Sun's NIS (Yellow Pages).

NIS maintains separate domains, each of which has a master server and multiple copy servers. While each NIS server can store the data for multiple domains, the data never merge. A client has to “bind”, or attach, to a particular domain on a particular server and can query data only in that domain.

SDDB also maintains separate domains, each with its own master server and copy servers. Each master server receives record updates for its own domain and propagates these changes to all the other servers across all domains. The data from each domain is merged on each server into a single contiguous database—the original domain being stored on each record. Thus, when a client queries the data, it does so across all domains.

The records are held as a “field=value” list of variable lengths. Only the values defined are stored in the record, and new values can be added at any time.

Indices are held in memory using a “red-black” tree algorithm. All creation and comparing is done by user-supplied functions, so the indices are very flexible. SDDB allows for multiple indices and can detect and reject duplicate entries, unlike NIS which allows only one index on each file or table.

The SDDB servers are completely stateless (i.e., they do not store any information between client requests) and use a fast UDP protocol to perform all transactions. A modification sequence number (which is analogous to a modification time) is held on each record so that a master server can decide what records have been updated and need to be propagated to the other servers. Since only the modified data is transferred, the propagation delay can be made very short—it is currently about 30 seconds.

SDDB has an API for both C and shell scripts. Thus, you can use either script to inquire, update or delete records in the database. The database is not tied to the print system—it can be used to store any sort of record-oriented data.

Effect of SDDB

I installed SDDB on every print server and converted all the master configuration data into SDDB records. SDDB could now provide the configuration data for mkprint, which produced the configuration files (/etc/printcap, et al.). I re-wrote mkprint in C (it was originally written in shell and awk script), which improved its speed enourmously. It no longer had to use rcp, since the data was already present on the local server. I rewrote the web (CGI) programs so that they no longer relied on the output of mkprint, and received their information directly from SDDB.

I wrote a front-end for SDDB, called **pradmin**, designed for the print system. It uses a simple command-line interface, similar to the Cisco router interface. Now multiple users can update the database simultaneously without fear of clashing.

As more and more programs came to rely on SDDB and the data it contained, SDDB became the glue that tied all the print servers together. A single update would affect many servers, which would all act in unison. Every print server knew about every printer at Cisco and acted accordingly. The “Distributed Machine” had started to take shape.

Linux Goes into the Field

Cisco started a spree of buying up small companies, particularly in the San Francisco Bay Area, so it was time to start installing more print servers. Linux machines were the best choice, since they are cheap. A Linux print server would cost under $2000 US, less than a third of its commercial rival, Sun's SPARC 5.

I ported and rewrote the remainder of the programs that, until then, had worked only on SunOS. Now the Linux machines could perform the full function of a print server.

A print server was installed a few miles down the road in Scotts Valley. Aside from a few teething problems, it worked. We then shipped one to Sydney, Australia. I preconfigured it with an IP address, so the only thing the system administrator in Sydney had to do was hook up the power and the network. It worked flawlessly. The SDDB server came up, copied its data down, I ran an mkprint and off it went.

rdist

rdist is a tool which allows the mirroring of directories between servers. This can be done either by doing an exact mirror (including the deletion of files) or just adding to and replacing directories already present.

Using this tool, we could put together a directory structure on a master distribution server and have it mirrored throughout all the other servers automatically. This was yet another massive timesaver, as the number of servers we managed increased. We even included the /etc/passwd and /etc/group files among those files updated using rdist.

Setting up a new server became straightforward: install a vanilla Red Hat release, then rdist the print system on top.

New Web Stuff

I did another rework of the web pages. I added the ability to stop and start print queues as well as delete a print job and send a test page.

I discovered that while using SNMP, I could display an HP printer's front panel display, which greatly aided in fixing run-of-the-mill problems such as “Toner Out”.

The web interface became the preferred tool for diagnosing printer problems and was made available to everyone. It allowed users to fix many of their own printer problems and not have to call us.

Failover

As I mentioned before, loccodes provided a rudimentary failover procedure. However, it would have been an arduous task to update all those loccode and sgroup records to denote a new server in case of failure. Luckily, I have never had to use it.

I added a backup server field to the loccode and sgroup records and created a new SDDB table called pserver. In it was detailed, among other things, the state of the server (“up” or “down”). I changed mkprint so that when it created the configuration files, it would check the state of the primary server, as designated on each loccode and sgroup record. If a server was marked as “down”, mkprint would direct LPR to use the backup server instead.

Now, failover was simply a matter of marking the dead server as “down” in SDDB and re-running mkprint on all the servers.

A Back End for LPR

The only problem with the mkprint system was that any updates to SDDB records were not instantly reflected in the print servers. Could I make the various programs bypass their configuration files and read SDDB directly?

I pulled open the LPR code and again replaced the routines that read the /etc/printcap file. This time, however, they read an SDDB record and created an in-memory printcap entry accordingly, using the same algorithms that mkprint used. I then did the same thing with BOOTP. It no longer needed to read /etc/bootptab, but rather read its information straight from SDDB.

Now, whenever changes were made in SDDB, print queues were created instantly and the BOOTP server was immediately available to service the BOOTP request. Failover didn't even require mkprint to be run. Samba was the only program left that didn't read its information directly from SDDB.

Linux Takes Over

After an extreme power outage at the San Jose headquarters' data center and general lack of performance due to capacity problems, the visibility of the print system was raised. Suddenly, managers realized that without the print system, nothing got printed and production lines stopped. The order came down to “double the capacity of the print system immediately”. I was given an assistant, Ben Woodard (bwoodard@cisco.com) to provide a much needed extra pair of hands. Although originally employed by Cisco as an MS Windows support line technician, within a few months I had converted him to a die-hard Linux fan.

We installed ten new Linux servers in the San Jose headquarters' data center. Linux print servers were spread around the world:

- 13 at the San Jose headquarters
- 3 around Silicon Valley
- 4 in Europe
- 2 in RTP, North Carolina
- 2 in Tokyo, Japan
- 1 in Sydney, Australia

On August 1, 1997, we retired the SPARC 20s and the other Axil machines. We are now completely reliant on Linux servers—a single line of commercial code cannot be found. We have (or at least have access to) the source code for every program we are running.

The Future

We have recently started replacing the print function in the Cisco branch offices. Until now, they have used local NT servers (with their associated problems) to manage their printers. Although the branch offices only have 5% of our printers, they account for about 50% of our calls. We have (as of February 1998) deployed 30 Linux servers to these branch offices, with many more on the way—there are currently about 200 branch offices in all.

There is still plenty of work to do as we expand the system to meet the needs of our printing clients. We have the following goals in mind:

- Improve SNMP features. Enable traps so the printers tell us when they have a problem.
- Create print queues using Java. The creation of print queues still requires someone who is happy to log onto the system and run pradmin, the printer queue administration program. Why not allow anyone with a browser and the right authorization to create queues?
- Replace the LPR program with LPRng or even an implementation of the Internet Printing Protocol (IPP). LPR is showing its age and does not provide many facilities (sending printing options such as **duplex**, for example).
- Extract page counts from the printers, using SNMP. We should be able to schedule regular maintenance visits by engineers.
- Implement a DHCP server. Many printers now support DHCP as well as BOOTP. A DHCP server would allow us to avoid allocating IP addresses to printers.

- Work on SDDB in order to make it usable as a general purpose Network Directory Service. I also need a new name for it.

Conclusion

Using regular UNIX tools and SDDB, we have created a distributed machine with multiple facets in the form of physically separate servers throughout the world. There has been no magic in doing this. It has been accomplished simply by fixing problems in a general way with an eye to the future. Linux has proved to be quite capable of holding its own in this large, “mission-critical” environment. Nothing is stopping multiple Linux servers from providing “big system” functionality and manageability.

For more information or to download the source for the code discussed in this article, go to http://www.tpp.org/CiscoPrint/.

Damian Ivereigh has been a Unix System Administrator and engineer for 12 years. He is originally English, but has been living in the USA for 4 years of which the last 3 have been at Cisco. He is a caver and a white-water kayaker in his spare time. He welcomes your comments at damian@cisco.com.

Archive Index Issue Table of Contents

Advanced search

Migrating to Linux, Part 3

Norman M. Jacobowitz

Issue #54, October 1998

The future of Linux in the SOHO environment.

Welcome to the third and final installment of our three-part series, Migrating to Linux from a Commercial Operating System. In the first article, I discussed some of the reasons why many people are switching to Linux as their desktop OS of choice for common, day-to-day tasks. The second one covered a few of the essential tasks that the small office/home office (SOHO) user of Linux should be familiar with, such as backups, emergency disks and other administrative tasks. In the final installment of this series, I'll take a look at what the future holds for non-technical end users who choose to migrate to Linux as their desktop operating system.

First, let's take another look at perhaps the biggest problem with using Linux as a desktop OS—the fact that Linux is often perceived as a “hackers only” OS. Linux is commonly regarded as a rough-hewn, not ready-for-prime-time toy, made by and for geeks. That image scares away a lot of potential end users, especially those whose first exposure to Linux may be from techie friends or from people in the IS department where they work.

For a significant portion of the history of Linux, that image was basically true. Linux had little to offer the end user. Just dialing into one's PPP account was a pain, let alone trying to configure the X Window system to one's taste or writing a complex document for a demanding client. Now, as simplified installation and more user-friendly GUI tools are becoming available, that is no longer the case. Yet the aura of Linux as a hackers' clique continues to some degree—perhaps due to inertia or the inherently decentralized method of disseminating information within the Linux community.

What will it take to dispel that myth once and for all? There is no simple answer. Short of a very well-funded marketing blitz and concerted information campaign in the popular media (not likely), it will not happen overnight. Within the Linux community is a lot of debate as to what constitutes effective advocacy. It is beyond the scope of this article to address each and every possible method of advocating Linux and open source software in general. However, I do feel there are a few things that every Linux end user can and should do to represent Linux and tout it as a viable desktop system for SOHO users.

Your successful use of Linux is the greatest advertising. In other words, if you are successfully using Linux, let people know. If you are ever asked what version of MS Windows you are using, be frank—explain that Windows does not suit your needs, so you are using Linux. Usually, that statement is followed by something to the effect of “Linux? What's that?” That's your opening to describe—or if you are in front of your Linux box, show—to them what Linux is and what it does. Whenever I go through this speech, I usually get a response like “but I am okay with Windows”. For some people, that's true. But a gentle reminder of how many times they end up rebooting Windows every week, day or hour may prompt them to ask more questions. On the other hand, don't be afraid to discuss some of the trials you have been through with Linux. No one is going to believe you if you present Linux as the Holy Grail of the computing world. Linux has its flaws, too. The difference is that there is an international cadre of conscientious programmers out there working to smooth out these bumps in Linux.

Support those who develop the software you use. It is important that those who develop software are rewarded in some capacity. Note that this applies to Open Source packages as well as commercial products. For many free software programmers, the reward is in the doing. Commercial software manufacturers are in it to earn money. Whatever the rationale and reasoning of the developers, they deserve support. If you find a piece of Open Source/freeware useful, let the developers know. Provide them with a bug report or a well-thought-out feature request. Let others know you are using that software, and share the URL of where to find it. If you are using a piece of commercial software, make sure you have registered the product and paid the licensing fee. Then, send any comments or constructive criticisms to the publisher. This lets the publisher know that their product is used and appreciated, and it may lead to more companies porting their products to Linux.

Help others migrate. Here is perhaps the best way to further Linux in the SOHO market. By helping newcomers get a Linux system up and running and helping them find the software tools they need, you'll be directly expanding the installed base of Linux. This may sound elementary, but helping another end user get a Linux box up and running requires a bit of time and effort. When I first dove into Linux, I wasted hours looking for solutions to problems that another user with just slightly more experience could have solved in a matter of minutes. Offer your best advice when asked, and volunteer to help if needed. That's why Linux exists in the first place—thanks to the combined efforts of a volunteer crew of users and programmers.

If we all take the above steps (at the very least), we will continue to see a growth in the number of end users who turn to Linux as their desktop system of choice. Perhaps the most impressive action that will lead to increased acceptance of Linux by end users is the refinement of Linux distributions and wider marketing of their products. By making distributions more stable, easier to install and easier to understand, commercial Linux developers are greatly enhancing ease of use for end users of Linux. Wider marketing means more exposure and a less mysterious aura surrounding Linux. These days, you can stroll into several mainstream software retailers and bookstores and walk out with the world's most powerful PC operating system for under fifty dollars (US).

Still, as a SOHO Linux user, I notice two major weaknesses working against Linux as a SOHO operating system. Number one is the support issue. While the Linux community is famous for its voluntary support resources, we still must contend with what appears to the end user as a lack of support. Although I don't claim to be an expert on software marketing, I have noticed a pattern in the way end users perceive the support services of software publishers. Basically, it seems to me that few consumers have much respect for the support behind any software, let alone free or Open Source products. People do care about technical support, especially in the early stages of installing and using a product. Since consumer expectations are very high and their perceptions of services provided are already very low, any impression of a lack of support will push many potential Linux consumers into an “I can't buy that” mindset.

The second major weakiness is the limited availability of end-user applications. While Linux is famous for native ports of best-in-class software like Apache, few, if any, desktop productivity applications are available for Linux that you can honestly say are the best on the market. The recent decision by Netscape to free the source code to its Communicator 5 product, and the decision by Corel to port their productivity applications to Linux, may show more of the corporate world that faith in the future of Linux is well-founded. Again, the Linux community is working against appearances and perceptions rather than realities—on paper, it looks as if the available pool of SOHO productivity applications for Linux is rather dry.

There is much talk about what will be the single (or few) most important “killer application(s)” that finally push Linux into critical mass in the network server market. Oracle, Sybase and names of other large-scale products are often mentioned. (Oracle and Informix have both announced ports to Linux.) The issue is a bit different when we are considering “killer applications” for the SOHO market. Most likely, no single product will make Linux more viable as a desktop solution; instead, it will be the collection of productivity tools that makes more end users feel comfortable enough to abandon their commercial OS. Right now, the compendium of end-user tools is small, but growing.

Now, I guess one of the key questions to resolve is this: how can we SOHO end users of Linux do our part to solve the support issue and encourage the development of more desktop productivity applications? Honestly, I don't know of any “silver bullet” answers.

The support issue can be addressed by following the three suggestions made earlier, along with some of the advice from the second article in this series. Eventually, I believe we need to see dramatic improvements in several key areas, including but not limited to documentation, the SOHO end-user support network and more easily installed and configured software tools.

As for documentation, it's easy to see why programmers hate doing it—it's tedious and may seem peripheral to the real goal of the project, which is producing the software. However, the documentation is often the key to selling the software (whether commercial or Open Source) to potential users. How many times have you installed a software package, and then been left on your own to figure out how to use it? If it means users of free software have to pay for third-party documentation of the program, then so be it. Remember, the key to success for the SOHO user is usability and productivity—whatever gets us there reliably and inexpensively is good.

When thinking about the SOHO end-user support network, it is tempting to point out the obvious—there is plenty of help available to end users in the existing support resources. While that is true, it is also quite tempting to imagine a scenario where SOHO Linux users have his own support network. Perhaps we need a new newsgroup, mailing lists and web sites devoted to our needs.

Making software easier to install, configure and use is a mantra being chanted throughout the software development world, regardless of the OS. One very exciting aspect of the growing Linux community is that more Open Source developers are turning their skills and attention toward end-user tools. The GIMP and KDE may be the best current examples. Such a combined resource pool of programming talent will naturally produce a host of higher quality products than can be achieved in the corporate world of tight deadlines, limited resources and expectations of marketability. I predict that the next few years will see a wealth of extremely high-quality, Open Source software tools for the SOHO end user.

Can we as SOHO users depend on Open Source developers to produce all of the software we need to get our work done? Eventually, maybe so. Right now, some of our needs would be best met by porting existing software to Linux. Personally, I have a dozen or so commercial software applications on my list of things I'd like to see ported to Linux. Every SOHO user can probably name a few of their own “dream ports”. How can we encourage the porting of software? Number one is to follow our earlier point of rewarding vendors who port their software. Positive feedback and payment of licensing fees will encourage developers to invest the time and energy in porting their products to Linux. Another approach is to visit the Linux Resources “Wish List” and suggest software you'd like to see ported to Linux. Your input on the page (http://www.linuxresources.com/wish/) will help the Linux community encourage more vendors to port their useful software to Linux.

While this series has been far from exhaustive, we've taken a stab at discussing the issues a SOHO end user needs to consider when migrating to Linux. The SOHO end user market is not the first niche one thinks of when examining how Linux will fit into the future of computing, but the issues of stability and quality that make Linux the best server platform also apply to the desktop.

Are you thinking of migrating to Linux for your personal desktop needs? I hope this series has helped you make up your mind. Have you already begun the process of migrating away from MS Windows or another OS? Then maybe some of the experiences shared herein can be of value. Are you a software publisher or vendor who is examining the future of Linux as a potential SOHO environment? Please rest assured there are non-technically oriented end-user types who see in Linux an alternative to poorer-quality commercial operating systems. While we may not be pioneers in the same vein as Linus Torvalds, we are on the cutting edge of adopting a better, more reliable way of getting our daily tasks done. Good luck!

Norman M. Jacobowitz is a freelance writer and marketing consultant based in Seattle, Washington. Please send your comments, criticisms, suggestions and job offers to normj@aa.net.

Archive Index Issue Table of Contents

Advanced search

Sculptor: A Real Time Phase Vocoder

Nick Bailey

Issue #54, October 1998

Sculptor is a set of audio tools for Linux that manipulates spectra in real time and provides continuous audio output.

Computer music in some respects places extreme demands on operating systems, especially now that inexpensive desktop platforms have enough raw processing power to perform relatively complex signal processing tasks in real time. Shared memory and System V IPC are powerful allies in realising audio manipulation tools under real-time control. Sculptor is a set of tools for Linux which uses these techniques to produce impressive throughput, even on modest platforms. It was initially conceived as a research tool, but may end up being a musical instrument.

Computer Music and Real-time Digital Signal Processing

This is the story of how a program, which ran in batch mode on PDP-11s taking many hours to produce a few seconds of audio output, can now be run in real time on an inexpensive desktop Linux machine. Changing a program from batch mode to real time presents an enormous challenge to the programmer: the user interface becomes an issue, imposing a completely new structure on the software, as user-interface-originated events need to be processed asynchronously with the real-time audio synthesis.

Timbre means sound colour, a perceptual correlate of harmonic content, in the same way that pitch is related perceptually to frequency. A violin and a flute can be played at the same pitch and loudness, but always have different timbre. One process which computer-musicians like to use is morphing, where sound can be altered smoothly from an initial timbre to a finishing one. Many readers will be familiar with the process of video morphing and will appreciate how it is an entirely different process from simply cross-fading. One method of achieving the audio equivalent is to manipulate the audio signal not as a series of time samples, but as a series of evolving spectra. By changing the attributes of the sound's spectrum as it evolves, this and many other interesting effects can be made.

In this article, a method for manipulating spectra in real time and providing continuous audio output will be examined. An example **xview** application has been written, so anybody with those libraries and an appropriate sound card can experiment for themselves.

Spectral Manipulation with the Phase Vocoder

The phase vocoder is one of the more powerful methods of manipulating sounds in the frequency domain. It is not a new technology; MIT's **CSound** application (see Resources), which was ported to the C language and UNIX from the original MUSIC11 program written in assembler for the PDP-11 minicomputer by Barry Vercoe, contains phase vocoder software. However, the algorithm was of such complexity and computers at the time were so short of processing power it would often require many hours of processing to realise each second of audio output. Only recently has sufficient processing power reached the desktop to make real-time phase vocoding a viable proposition.

A vocoder is an electronic signal processor consisting of a bank of filters spaced across the frequency band of interest. Originally, it was hoped such a device would be able to reduce the bandwidth necessary for the transmission of voice telephony, but it rapidly found other applications in popular music. A voice signal could be analysed by the filter bank in real time, and the output applied to a voltage-controlled filter bank or an oscillator bank to produce a distorted reproduction of the original. The effect can be heard in some Electric Light Orchestra tracks, and in the theme music to the film *Educating Rita*.

After Michael Portnoff (see Resources) demonstrated an efficient method of building the required filter banks digitally, the door was open for a computer-based implementation of a digital phase vocoder, bringing with it a vast number of possibilities for the analysis, manipulation and synthesis of audio. Wishing to use this technology to improve my understanding of the relationship between a sound's timbre and its spectrum, I set about writing Sculptor, a real-time and interactive phase vocoder for Linux.

Implementing a Phase Vocoder Digitally

The Phase Vocoder in Sculptor comes in two parts: a batch-mode analyser called **analyse**, and a real-time synthesiser called, perhaps more imaginatively, **prism**. **analyse** reads an input file in Sun/NeXT audio format. The sample rate we use most often is 22,050 samples per second, as my P120 machine at home can comfortably keep up with this resynthesis rate using floating-point arithmetic with enough power left over to see to the work of running the X

Window System interface. Samples can be acquired in the usual way using a command-line recording tool, but finding that rather tedious, we wrote **Studio** (see Resources) in Tcl/Tk to make the process of acquiring short samples more accessible.

Figure 1. Analysing the Audio Samples

analyse reads the sample file and breaks it up into overlapping windows of about 10ms in length. This window length was chosen because evidence suggests the ear is insensitive to spectral changes on a shorter time scale. Each window is Fourier-transformed, producing an array of spectral samples (see Figure 1), but instead of simply storing the amplitude and phase of each Fourier result (bin), the amplitude and phase change per window are recorded.

To understand why the phase change per window is important rather than the absolute phase, let's consider a simple example. Suppose we are using a sample rate of 8192Hz and have a 128-point FFT. Each window will last approximately 15ms, and the spacing between Fourier bins will be 64Hz. Now present this program with a sine wave at 1KHz. The Fourier transform cannot represent this signal exactly; recall that it is behaving like a bank of filters 64Hz apart, so the nearest filter frequencies will be bins 15 and 16 at 960 and 1024Hz, respectively.

When this same signal is analysed a quarter of a window later, it will still be represented as a 1024Hz sine wave. Since its frequency is actually lower, it will appear to have lagged in phase. By storing the phase change per window, sufficient information is retained to at least approximate the original 1000Hz sinusoid by overlapping the inverse Fourier transformation results and adding them together at resynthesis time.

Real Time, Not Real Fast

There is a big difference between a sound-synthesis program which runs in real time and one which simply produces output samples at a greater rate than the sound device chooses to swallow them. For a program to be a real-time synthesiser, it must respond apparently instantaneously to a change in an input parameter. For example, the CSound application mentioned previously is not real time, because it reads the specification of a score and orchestra at initialization, then produces audio output. It isn't possible to influence the sound the program produces as it produces it. (Actually, some real-time extensions have become available, but I am choosing to ignore them for the sake of this example.) Running CSound on a powerful workstation usually causes it to produce samples faster than actual speed, but this does not qualify it as real time.

To design a real-time program, one of the most important design considerations is the user interface, which in turn is strongly influenced by the desired effects. The next stage in the design process is considering the kind of manipulations required for such an application.

The Phase Vocoder as a Musical Instrument

When a synthesis program becomes real-time, it becomes a musical instrument, and when a computer program becomes a musical instrument, the operator becomes a performer. The ergonomics of a musical instrument are highly complex, but from the context of previous uses of this algorithm in computer music, clearly some core areas must be covered: pitch transposition, a change in a sample's pitch with no change in its duration; rate of playback, a stretching or compression of a sample in time with no change in its pitch; and timbral morphing, where one sound changes smoothly into another as pictures do in video morphing.

Sculptor permits independent control over pitch and rate of playback in real time even on very modest computing platforms, and acts as a test bed for more advanced algorithms on faster platforms. It was initially developed on a 386DX40 without a floating-point coprocessor and could make a fair attempt at real-time synthesis at 8000 samples per second (voice telephone quality).

Choosing a Paradigm

Having decided there are essentially two parts of the application, a real-time synthesiser and a GUI, it seems to make sense to divide the processing between the two. One process will be responsible for the audio synthesis, the other for mouse- and window-related processing.

Linux, like most UNIX systems, provides two different methods for inter-process communication (IPC). The first is channel-based: sockets, pipes and so on. This kind of IPC has many advantages; one can easily map the processes onto different machines connected by a network, and synchronization is easily arranged, as a channel can be set up to block in an efficient, non-polling manner until data arrives.

The prism application has two processes which basically operate asynchronously. Essentially, the resynthesiser has to keep running and producing audio samples regardless of what the user is doing with the mouse. For this reason, the second method of IPC, shared-memory or System-V IPC, has been used. System-V IPC also provides methods for process synchronisation: the semaphore. One can raise or wait on a semaphore. Think of it as a special kind of variable which behaves in the following manner. If one or more processes are waiting on a semaphore, raising it enables exactly one of those processes to proceed. If no processes are waiting, then the value of the variable is incremented. Waiting on a non-zero semaphore decrements its value but allows the process to continue immediately. Waiting on a zero-value semaphore adds the current process to a (possibly empty) list of waiting processes, pending the semaphore being raised by another agent.

Semaphores are used in shared-memory situations to implement mutual exclusion locks and prevent update anomalies where several processes simultaneously attempt to modify a shared data structure. However, prism uses only two processes accessing the shared-memory block: the GUI is a producer because it is supplying control parameters, and the synthesiser is a consumer because it uses them to generate audio samples. Since there is only one producer and one consumer, there is no need to use semaphores as access arbiters. In fact, advantage is taken of the shared-memory IPC to allow the producer to provide a set of “magic” parameters which change according to the user's gestures.

Allocating and Using Shared Memory Blocks

Upon startup, prism has to allocate and set up a shared-memory block, then fork off the process to generate the audio output. The routines it uses are documented in the **shmop** manual pages. Enough memory is allocated to hold a control structure and all of the spectral data produced by the analysis program (see Listing 1).

prism calls **shmget** to allocate the required amount of memory; it returns a handle to the memory block for subsequent use. The other parameters specify the access permissions in the normal **chmod** format, and the block will be created if it does not exist yet. The process then forks with the child being responsible for synthesis, and the parent for control functions.

After the **fork** call, both the parent and the child processes must attach the shared-memory block and cause it to appear in their respective memory spaces. The appropriate system call is **shmat**. The parameters indicate the handle of the shared memory block and the desired target address. Passing NULL as the latter tells the system to make the choice of address. In Linux running on an i386 architecture, the blocks are allocated downwards in memory starting at an address of 1.5GB. This call can be made once before the fork system call, as the shared block will then appear in both the child's and parent's memory space.

One trap waits to catch the unwary programmer using shared memory blocks: they are persistent. If your application crashes without properly tidying up shared-memory blocks, memory will leak like a sieve. The user can check for any undeleted memory blocks using the **ipcs** command and remove them with

ipcrm. **prism** does its best to cope with any unexpected events by catching the SEGV signal and shutting down any shared-memory activity before exiting. However, the best safeguard against memory leaks is to mark the shared-memory block for deletion as soon as it is created. Counterintuitively, the way to do this is to mark the block as transient using the **shmctl** call, and then detach the process from the shared-memory block. The shared-memory block will persist until all the processes using it detach using the **shmdt** call, so the block will disappear automatically when the parent and child processes exit.

Choosing a GUI

Writing Sculptor was supposed to be an exercise in efficient signal processing rather than sophisticated GUI design. The GUI library eventually selected was xview, which is a rather aged library, although dynamic and static versions are still available for Red Hat's 5.0 release based on glibc2, so it is not quite dead yet. The reason for this choice was primarily because I was familiar with it. The principal requirements of the application, a few simple menus, sliders and the detection of mouse events over a performance canvas are satisfied by practically any widget set from the oldest Athena to the latest GTK+, so the wide availability of the library in an open form is of more concern than its technical sophistication. *Linux Journal* has already published an article (see Resources) demonstrating the use of xview and the ease with which simple applications can be constructed using variable-argument-list calls.

When the prism application is invoked by the name **xprism**, the xview GUI is enabled. Most of the data flow in xprism is mono-directional: the GUI process produces control data, and the synthesiser consumes it producing audio samples. However, when the specification for xprism was conceived, it became evident that asynchronous flow in the opposite direction was needed.

When xprism runs, the following sequence of events takes place:

- The analysis file is opened and the size of memory required to contain the spectra is calculated. An additional allowance is made for spectral workspace. As in interactive mode, a mouse action might be responsible for setting the synthesiser parameters rather than relying on simply incrementing through the spectrograph automatically.
- The shared control block (see Listing 2) is set up with appropriate initial values.
- The whole spectral file is read into the shared-memory block.
- The process forks a child responsible for audio resynthesis.
- The parent xview widget hierarchy is intialised and the spectrograph drawn.

- The parent informs the child it may generate audio samples by setting the **xv_ready** field in the shared control block.
- The parent and the child run concurrently, using the shared memory area for data transfer (see Figure 2).

Figure 2. Data Flow in the Shared Memory Block.

This process is fine for simple gestural control, but as it stands, there is no feedback from the synthesiser process about whence in the spectrogram the resynthesiser is taking its data. Refer to the screenshot of the application running in Figure 3, and it is obvious that two fundamentally different ways of getting sound out of the program are present.

1. The user can select a playback speed from the speed slider, in which case the rate of advance through the spectrogram is under control of the resynthesis process. It is necessary to update the green line when the resynthesis source is moved automatically by the resynthesiser, and of course this happens asynchronously with the GUI events. The solution is to arrange for the resynthesiser to signal the parent when an update is required.
2. The user can click or drag on the spectrogram, in which case the spectrum to be resynthesised is under control of the parent controlling process. There is no problem in positioning the green line on the spectrogram canvas: it is simply dictated by the position parameter from the serviced mouse-motion event.

Figure 3. Application Screenshot

Using Signals in xview Clients

Sending a signal from the child process is easily achieved, but there may be unfortunate consequences. The application is continually sending commands to the X-server, and in principle, a signal might occur during the process of client-server communication. This is dangerous only in that the signal initiates X-server commands (in order to draw the green line), so some method of making X protocol requests atomic has to be employed.

Fortunately, the necessary methods are provided by the xview package. An xview client is not supposed to perform certain operations. One of these is servicing interrupts received directly by the signal method, and another is using sleep to suspend itself. Both potentially interfere with the proper operation of client-server communication. If the client wants to use interrupts, it must register itself as a signal acceptor as follows:

```
notify_set_signal_func(frame, update_frame_posn,
        SIGUSR1, NOTIFY_SYNC);
```

This call appears at the end of the initialiser for the xview data structures and associates the service routine for the signal SIGUSR1 with the servicing function **update_frame_posn**. The **frame** argument is the parent frame of the application, and NOTIFY_SYNC indicates that servicing of the interrupt should be delayed until pending X-protocol exchanges are complete.

Conclusion

Sculptor is a package that can manipulate sound samples in potentially exciting ways. It has a front end which permits the user to perform these manipulations in real time. System-V IPC has been used to split the process into two halves which can be efficiently load-balanced on a multi-processor machine. The source code is freely available (see Resources).

Sculptor is not intended as a guide to good programming practice; in fact, some of the code is just plain ugly. Early in its development, the whole application was required to work in real time on a 386 machine without a co-processor, so some sacrifice of clarity for speed had to be made. Nonetheless, it is hoped the code might stimulate other projects in real-time audio manipulation, now that one based on a processor-intensive algorithm has been demonstrated to operate fully interactively in real time. The application is also being used to support research into computer music at the University of Leeds, and so by definition will never be “finished”.

Resources

Nick Bailey (N.J.Bailey@leeds.ac.uk) obtained a B.S. in Computing and Electronics from the University of Durham, England. Having worked at British Telecom Applied Technology in West London, he returned to Durham to study for a Ph.D. in the application of parallel computing to audio signal synthesis. He is currently a lecturer at the University of Leeds in Applied Computer Systems at the Department of Electronic and Electrical Engineering, with additional responsibilities for Overseas and European Liaison. He enjoys old, unreliable, fast cars, and owns a cello, but demonstrates no discernible talent in either direction.

LJ Interviews Robert Dinse of Eskimo North

Marjorie Richardson

Issue #54, October 1998

Some background on the ISP and its switch to the Linux platform.

A lot of rumors have been circulating about the Internet service provider (ISP) Eskimo North (http://www.eskimo.com/) converting totally to Linux. I decided it was time to talk to Eskimo's owner, Robert Dinse, and find out which rumors were true. We talked via e-mail on July 9. Eskimo is a popular ISP based in Seattle, so Anne Tiemann went over and took the pictures shown here.

Margie: I understand you have just about completely moved to Linux on SunSPARC workstations. Is this true? What were you using before?

Robert: This is somewhat incorrect. We have seven Suns running SunOS 4.1.4, six Suns running SPARC Linux (Red Hat) and one PC running Linux.

In some areas Linux is clearly superior, and in some it still has problems. In general, it works well with SunOS, so mixing platforms isn't a problem and we use the OS best suited to any particular task.

Margie: What factors entered into your decision to switch to Linux?

Robert: With SunOS we are locked into one hardware platform, SPARC. Basically, SunOS is no longer being developed, and much new hardware is not supported by SunOS.

Sun's replacement, Solaris, isn't particularly fast in my experience, is buggy, bloated, and you can't do anything without checking with a license manager first. The license fees are ridiculous and the support from Sun has always left something to be desired. There seems to be a new root exploit almost daily for Solaris, and fixes from Sun are often far from instantaneous.

So continuing down the Sun path using Sun's new operating system was not a viable option for us. It's not cost-effective in terms of either the actual cost of the software or efficient use of hardware. Without source code, we are at the mercy of the vendor—a situation I wanted to get away from.

Linux, on the other hand, was available for a multitude of hardware platforms. Intel class CPUs are adequate for low-end applications, and extremely inexpensive relative to SPARC CPUs. For high-end applications, Alpha and Power-PC chips are beginning to look attractive. Suns still have some architectural advantages that may continue to make them the best hardware choice for disk-I/O-intensive applications. The main advantage is the fact that the peripherals are also mapped through the MMU (memory management unit), so that disk I/O can be a direct memory access (DMA) into user space instead of having to access kernel space and then be copied by the CPU into user space, although with today's faster CPUs, this is becoming a minor point. The point is, whichever of these platforms is best for a particular application, Linux gives us the freedom to use it. With SunOS, we can use Suns or a SPARC clone but not Intel or Alpha or Power PC, and while Solaris does support Intel, it's my feeling it is of marginal functionality on any platform because of Sun's perceived need to enforce arcane licensing policies with license managers.

Full source code is provided for Linux, so we are not at the mercy of a vendor for fixes. If we have any questions about exactly what the code is doing, we can just look at the source.

Linux is still being developed, which means that as things like IPv6 become mainstream, it will remain viable where SunOS will not. The software isn't straddled with having to wait for a license manager to say it's okay for another user to log in or to use another instance of a particular resource, because there is no license manager. This, in my opinion, is a major problem with Solaris.

Margie: What is the principal use for your Linux machines?

Robert: We have a Sun SS-10 with quad Ross RTK-625s and 256MB of RAM running SPARC Linux that serves as our main web server, http://www.eskimo.com/.

We have these Sun LXs running SPARC Linux that serve as our client IRC server irc.eskimo.com, our HUB IRC server hub.eskimo.com, and our virtual domain web server www2.eskimo.com, and one that serves as our FTP server ftp.eskimo.com and our adult web server adult.eskimo.com. The IRC machines are equipped with 64MB of RAM, the other LXs are equipped with 96MB.

We have a Sun 4/670MP with dual Ross RTK-625 CPUs, 256MB of RAM, and a second SCSI controller running SPARC Linux that serves as our news server eskinews.eskimo.com. We actually have two more CPUs for this machine, but a bug in SPARC Linux prevents it from working with four CPUs on the 4/670MP platform.

We also have a Pentium-based PC running Linux that is one of our mail servers, mx1.eskimo.com (external/list). Additionally, name servers run on some of these machines.

Figure 1. Stack of Suns running Linux

Margie: How about your “not Linux” machines?

Robert: Our main interactive host, eskimo.com (where shell accounts and web pages are located), is a Sun 4/670MP with quad Ross RTK-625 CPUs, 384MB of RAM and an FDDI (fiber distribution data interface), so it sits directly on the dual FDDI network backbone. In addition to interactive shell accounts, it also provides all the user file space.

We have a Sun IPX with a Wytek Power-uP 80 MHz CPU and 128MB of RAM that is used for SLIP and PPP emulation programs (tia and slirp), tia1.eskimo.com; a Sun IPX with 64MB of RAM running as a mail server (external/list), mx2.eskimo.com; a Sun 4/330 with 96MB of RAM which runs IRC bots, chat.eskimo.com; a Sun 4/260 with 108MB of RAM that runs a Moo, name server and empire game, isumataq.eskimo.com; a Sun LX with 96MB of RAM and FDDI that is the mail server hosting the spool directory, pop server and internal SMTP server (the one customers connect to internally as opposed to one that receives external mail); and an IPX with 64MB of RAM that is used as a work station.

Margie: Will you be changing these machines over too? If not, why?

Robert: Yes and No... Eskimo, Tia1, Isumataq and Chat will remain SunOS until such time as Linux is more predictable from a security standpoint, since they have interactive users logging into them.

Mail and Eskimo will remain SunOS until FDDI support for the FDDI cards we use is part of SPARC Linux, or until we find an affordable solution to interface 100-base-T which is supported by SPARC Linux to our dual FDDI backbone.

The mx2 mail server may be converted to SPARC Linux in the not too distant future. Earlier versions had serious problems with NIS, but those have been fixed in glibc, paving the way for the switch.

Earlier versions of Linux also had problems with Sun4c architecture, so the one IPX we are using as a workstation still runs SunOS. We haven't had time to experiment with recent SPARC kernels to see if that issue is adequately resolved.

Margie: What do you feel are the main advantages for an ISP to use Linux?

Robert: The main advantage I suspect is driving most ISPs is cost. It's hard to beat free, not counting the cost of media and support offered by various vendors.

We also found some technical advantages. SunOS libraries support only 256 file descriptors per process. We needed more than this for our web servers and IRC servers, so going to a different OS wasn't an option for these machines. Mail servers run much more efficiently with more file descriptors, because you can use a larger open connection cache, and less thrashing (closing and opening of descriptors) is necessary for operations such as list processing.

For News, SunOS is limited to 2GB file partitions. This is completely inadequate for news volumes today. While SunOS and Solaris have something called “Online Disksuite” available, it is at best a kludge to get around a fundamental problem with the OS. Linux's support for huge stripped partitions made it attractive for the news server.

Last, Sun hardware has not remained cost-competitive with other platforms, and we wanted the freedom to migrate towards other hardware platforms such as Alpha. Linux provided this freedom.

Margie: How about disadvantages? Tell us about any problems you have had.

Robert: NIS and NFS bugs. NIS would fail randomly with the older libs, making it unusable for applications where proper user lookups were essential, such as mail servers that need to deliver mail, or web servers where a user's directory has to be located. We worked around this on some machines, where the file descriptor issue was important, by putting together a kludge to place a local copy of the password file on the Linux boxes that had broken NIS libraries. This solution is inefficient. **glibc** fixes this problem. However, on SPARC platforms no compatibility library is provided, so all applications must be recompiled. This process is going to make conversion a somewhat tedious project.

NFS works, as long as the servers are alive. But if the NFS server goes away for some reason (is rebooted, for example), the NFS client machines will not always recover without a reboot. This isn't a complete showstopper, but it's very annoying.

While Linux supports a broader spectrum of hardware, it does not support all of the SPARC hardware periphery supported by SunOS, such as the FDDI cards or a 4/670MP configured with four CPUs.

Then there are security issues. SunOS is fairly static at this point, so most of the security problems are well known and fixes are available. Linux is still in a constant state of flux, resulting in new security holes being introduced—it's a moving target. Between that and the fact that I have more experience with SunOS, I feel more comfortable with SunOS where users are actually able to log in to the box.

Margie: What do you feel is needed to turn Linux into the number one OS choice for ISPs everywhere?

Robert: I'm not sure it isn't already. Most ISPs seem to use PC platforms, and cost is important. A percentage of them just don't feel comfortable without a vendor holding their hand, and in this area Linux is at a disadvantage. There are also really strange people who want to see Microsoft on the label and insist on running NT, even though they might need ten times as many machines to do the same amount of work. I do not understand the psychology of that group at all or know what Linux could possibly do to appeal to them without ruining Linux's functionality.

In terms of general improvements, glibc with its fix for NIS and some security improvements is a big win. If someone would fix NFS so it would properly recover after NFS server unavailability, that would also be a major win. Of course, support for hardware that we already have in use such as the S-bus FDDI cards and quad CPU configurations of the 4/670MP would be a plus for us, but I doubt it would be significant to ISPs in general, since Intel hardware is a more common platform these days.

Margie: I understand you have written software for a menu-driven user interface for ISPs. Have you ported it to Linux? If not, do you plan to?

Robert: The software I've written is for Eskimo specifically. It is not something I intended to sell as a software product, but rather intended as a feature of Eskimo. Actually, at this point, it's pretty antiquated. I wrote it before the Web existed, and in modern times, the multimedia capabilities of the Web really provide an opportunity for a much better interface. So I have no plans to port the old software to Linux, though newer software I am now working on is web-based and native to Linux (and probably not portable to other operating systems because I am developing it for our own use, not for resale).

Margie: Tell us a bit about Eskimo—some history, customer base, etc.—blow your own horn.

Robert: Eskimo North started as a single-user BBS in 1982, and its initial creation was somewhat accidental.

In my junior high and high school years, I was involved in pirate radio (broadcasting without an FCC license). I had a number of friends who were also involved and ran their own pirate radio stations. Several of them were busted by the FCC, and one of them was fined $750. I narrowly escaped fines after my own station caused some interference in the 80-meter ham band and drew the attention of the FCC. They sent me a cease and desist order, though it was specific to the interference caused to 80 meters and said absolutely nothing about the illegal broadcast in the AM broadcast band. I had my first-class radio telephone license by this time and didn't want to lose it, so that order ended my direct involvement in pirate radio. However, I did experiment with various legal alternatives such as carrier current and getting the absolute maximum signal possible under the limitations of part 15 rules and regulations for unlicensed transmitters.

One of my friends started a newsletter dealing with pirate radio after he was fined, and since I couldn't be directly involved safely anymore, I became interested in this. The way they were putting it together was to type it up on an old Royal typewriter, cut and paste it and make photocopies at a 7-11. I thought a computer and printer might be a more efficient method and purchased both, in part for that purpose but also because I was interested in computers.

I hooked up a modem but there was no built-in support for the serial port in the machine (Radio Shack model III); so I wrote what was initially a primitive host program. At first, no login was provided and the number wasn't advertised. People I wanted to allow access to were given the number, and they could call with a modem, invoke a word processor and do what they needed to do.

Even in 1982, people were running war dialers that found the number and they started messing with the files, so I put up a primitive BBS front-end. Its primary function was to keep people out of files they didn't have permission to. I developed the host program.

At that time, Glenn Gorman was running Minibin, an early “room-based” BBS, written in BASIC. Glenn wanted to turn Minibin into a commercial product, but it was written to use a host program specific to the Microperipheral bus decoding modems. At the time, a serial interface was not standard, and so Microperipherals Corporation designed a modem that connected directly to the bus, negating the need for a serial card, for around $300—this made it a popular choice. But, Glenn couldn't sell Minibin using their drivers because they were proprietary, and they designed their modem so that the driver would work only with their modem. So he approached me with the idea of using my host software with his BBS software. However, my approach was much different than Microperipherals, and getting my host program to work with Glenn's BBS required fairly extensive modifications to both.

During that time, I ran a Minibin replacing the software I was running. But, Minibin at the time did not have support for private e-mail or file upload/download, both of which were things I had in my software, so I added that functionality. Having two Minibins caused no end of confusion; people didn't understand why when they called one, their login didn't exist on the other. They didn't understand that they were separate systems. A large number of BBSs running the same software was not common then, like it is today. So in an effort to clarify things, he named his BBS Minibin South, and I named mine Minibin North. People were still confused, so when he temporarily changed his to Jamaica South, I thought, hmm, what's up north? Eskimos are up north, hence Eskimo North.

Figure 2. Modems!

During this time, I had also come across some quad density double-sided 80-track drives and had a whole 3MB of disk space. I know that's nothing today, but back then when I had only 48KB of RAM and programs were typically quite small, 3MB was a lot!

By this time the BBS was well-known, and with the help of some of my users who were true machine-code gurus (I could do okay with the help of an assembler and disassembler), I had Infocom adventures on-line and a bunch of other games. For its day, it was fairly advanced. A book was published by an author in Alaska, and in the book he said, “Eskimo North ... a strange system. Must call at least once even if it's long distance.” With an endorsement like that, the name stuck. By 1985, I could pull the plug at 4AM on the modem, plug it back in, and immediately someone would connect.

Just prior to that, I had obtained a trial account on Compuserve, and I was impressed with all the cool things that could be done in a multi-user environment, but at $15/hour for 300 baud and $24/hour for 1200 baud, I could not afford it and I knew that was true for many others as well. Some other friends I had met through the BBS ran a network of four BBSs that were run on Apples tied together with super-serial cards that permitted something similar to MUDs, a multi-user D&D kind of environment, but more gruesome and more creative than any MUD I've seen since. The hardware was always breaking down and was generally difficult to maintain or scale.

I decided to go multi-user with Eskimo and started looking for a workable platform. Networked Apples didn't look like the answer. MPM looked capable, but I felt that the address space limitations and hardware dependencies would limit its future lifespan. Thus, I decided on UNIX; the only affordable UNIX system I could get my hands on was a Tandy 16B with an ugly, primitive version of Xenix. That was our entry into the UNIX world, initially with four phone lines.

We took that machine as far as we could, basically upgrading to a Tandy 6000 with 4MB of RAM, two 70MB disks and 11 lines.

In 1991, I bought a Sun 3/180, an ancient MC68020-based Sun that absolutely ran circles around the Tandy, and we upgraded it to a 3/280, then a 4/280, then we took news off the machine and put it on a separate 4/330, and we've been growing and adding new hardware and functions ever since. When the Internet became available, around 1995, we added Internet connectivity, initially with a 14.4K PPP link, then 28.8K, then a 56K frame relay, then a T1 frame relay, then a dedicated T1 to Sprint, now with dual T1s to Sprint. We also utilize a port wholesaler with T3 backbone for 56K dial-up access.

Margie: What do you feel the future holds for eskimo.com?

Robert: Well, there are two aspects to the Internet as I see it: access, essentially the roads that get you where you want to go; and hosting in all its forms, essentially the land, the buildings, the space where things are kept and where people meet and interact.

I will continue to provide access as long as there is a demand for it and we can address that demand in a cost-effective and efficient manner.

However, I am personally a lot more interested in the hosting side of things: what can be done with the Web and helping people create their own web space and make it available to the masses.

I personally do not share the vision of the Web as being a glorified infinite-channel cable company. I don't share the vision that so many have of making the Web a place where a few people control all the content and spoon-feed it to the masses. Rather, my personal interest is in getting the masses involved in creating content and sharing it and interacting with each other.

It's my view that for every problem facing mankind, there is probably someone somewhere who has an answer. The Internet has the potential for allowing those ideas to be heard, expressed and implemented. Additionally, while some fear the sharing of cultures will lead to a mono-culture, or essentially no culture, I believe quite the opposite is true. I think people will hang on to their own cultures, languages and traditions, and feel that the sharing that results on the web will lead towards more tolerance and understanding and an enrichment of all cultures. I want Eskimo to be a part of that process. I've tried to encourage people by providing good tools for web development and making disk space available cheap. We also have no data export fees, so users are not punished for creating a popular web site.

To this end, I've tried very hard to involve our customers in what we do. We have some local newsgroups, the most important of them being “lobby” which serves as a forum for our users to discuss things relevant to our operation. I share plans and ideas with our customers and solicit ideas from them.

Our customers have truly driven us to where we are today and while I'd like to say to you I have solid plans to make Eskimo this and that in five years, the truth is I intend to allow them to continue to determine our direction.

Margie: Anything else you'd like to tell us about?

Robert: While we've had customers from all over the world for some time utilizing host features here, until recently we've had dial access available only in Western Washington. We now have 56KB available in many metropolitan areas in the U.S. including shell access, web hosting, etc. I'm not sure how this is going to affect Eskimo's future, but I am going to try very hard to make it a place that people will think of as a home in cyberspace. Phone numbers for reaching us are 206-361-1161 and 800-246-6874.

Margie: Thank you very much.

Archive Index Issue Table of Contents

Advanced search

The Great Linux Revolt of 1998

Chris DiBona

Issue #54, October 1998

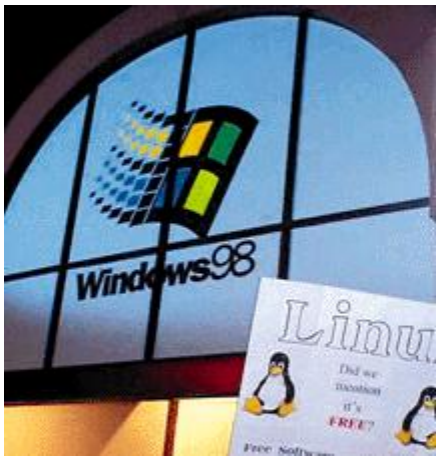
A new and fun way to positively promote Linux.

When Sam Ockman's notice hit Slashdot (http://www.slashdot.org/) at 6:21PM, it read:

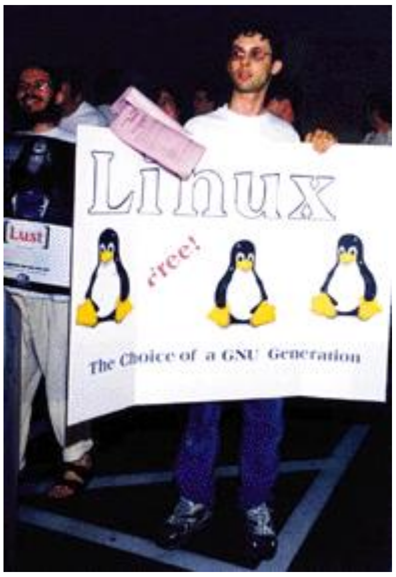
> The 500 members of SVLUG are going to have a big rally tonight at the largest/most prestigious computer store in Silicon Valley, Fry's, when they stay open until 1AM to distribute Windows 98. We're going to hand out Linux CDs and stuff like that as well. We'd like to have more people at the rally than go to buy Windows 98.

While it was obvious the entire membership of the LUG was not planning on showing up for a midnight rally, it was clear this was an idea with great potential. A little after 10PM, Sam and I arrived at the Chili's restaurant near the rally site to meet with people beforehand. We hoped we wouldn't be alone at the table.

We were not disappointed. There were about 15 people waiting for Sam to arrive. Some we recognized from the SVLUG, and others had seen the notice on Slashdot. Remember, only four hours had gone by since the original posting. By the time we left, the crowd had grown to 25 people.



The Fry Electronics display window—with a Linux sign in front of the dreaded logo!



Sam Ockman, SVLUG President

Press interviewing SVLUG Member

After consuming some munchies and a stout or two, we headed for Fry's Electronics in Sunnyvale. For those of you who aren't familiar with Fry's, it is a warehouse-sized electronics, software, hardware, test equipment, junk food, book and magazine retailer here on the West Coast. Picture Price-Costco, except with resistors. Most people have a love hate relationship with Fry's, mainly on customer service issues. It's important to note that this rally was not a protest against Fry's, but a rally for Linux on MS Windows 98's birthday.

We arrived at the Fry's store around 11:30PM. The 500 CD-ROMs from the generous folks at S.u.S.E. hadn't arrived yet, and we were a little nervous that they would be late. At Fry's our numbers had doubled to a little over 50 people. By now, our signs had been pulled out and we were actively agitating for Linux. This was a peaceful gathering; we made a point of not being too annoying and staying out of the way of cars.

The S.u.S.E. CDs arrived around midnight, coinciding with the beginning of the Windows 98 CD sale. We started handing them out along with pamphlets designed by Adam Richter (of Yggdrasil) to people who were coming into and out of Fry's. At this point we had two reporters show up to document the event (see Resources below).

Also at this point, we started noticing worried-looking guys with walkie-talkies watching us from the parking lot, and some Fry's managerial types talking into their cell phones. One of the protestors overheard one of their managers call the police, and then frustratedly say to another Fry's guy, “Why can't we just go over there and rip those signs out of their geek hands?” When we heard that, we were taken aback, but it did reconfirm our desire to continue.

A few minutes later, the Sunnyvale police began to arrive in numbers. Four police cars pulled forward into the lot. I offered myself up as their contact, while Sam controlled the Linux people.

The police officer asked us what we were doing there. I described Linux and how we were trying to get the word out about it. The police officer was cool about the situation, telling me he'd have to wait for his Lieutenant to arrive before deciding what to do. We were on Fry's private property, their parking lot. It turns out there are certain situations where it's okay to protest on someone's private property. It's pretty much always okay to protest on the sidewalk. We had been expecting to be asked to move to the sidewalk eventually.

Before his Lieutenant arrived, one of the other police officers asked more about Linux. The word is one of the Linux people got a CD into his hands on his way out of the lot. He did say, “Hey, good luck with your system” to me.

Additionally, the manager of Fry's came up to me (the same one who wanted to rip the signs out of our “geek hands”) and asked me why I was doing this, as Fry's sold Linux in their book section. I pointed out that we knew this and indeed had a sign showing people which aisle to go to for the book. He then asked why the Linux marketing people or my (as if I'm in charge of Linux!)

vendor relations people hadn't contacted Fry's management to arrange an event similar to the Windows 98 launch. Needless to say I was like a deer in the headlights. I told him that, in fact, Linux was a free operating system, and this was one way we saw to market it (not to mention the fact that no company, let alone a small user group, could match a MS launch when it comes down to the checkbook).

He was visibly upset, so I thought it was best to stop talking with him at this point. I told him any further communication between us would come to no good end, and we should only talk through the police. (His brains were melting out his ears by this point.)

The Lieutenant arrived and said we should move to the sidewalk, so we did, and it was just as well; cars were actually pulling over and picking up a copy of the Linux CD.

After about 15 minutes of this, we decided to hit the CompUSA store a few miles away. When we arrived, there were still a lot of people in line there. It turns out CompUSA was not only promoting Windows 98, but also doing all kinds of things to bring people into the store. They were selling computers for $98 to the first 10 people in line, and paying people $4 to take away 32MB of RAM. (Actually, the RAM was $32, but there was a $38 rebate—or something like that, it was weird.) This also meant there were a lot of people to promote Linux to. CompUSA's management people were cool; they took the rally in stride, telling Sam the rally was okay, as that was what the First Amendment was all about. Fry's had sent one of their security drones to CompUSA to warn them of our existence, but CompUSA just let us do our thing.

After CompUSA petered out, we converged on a local Denny's to nosh. Denny's flipped when they saw all the people, so we moved to a braver restaurant down the street. I got home around 3AM to find that people were already uploading pictures to the web.

Crowd at Fry's

Rally Members at Fry's

By the end of the night, we had given out 500 CDs and hundreds of pamphlets. A total of six press people showed up and there were over a half-dozen mentions in major newspapers. This is not counting the coverage we received from other on-line sources, such as CNN and the BBC. Everyone involved had a rocking good time promoting Linux.

Whether or not you agree with this direct-action model of Linux marketing, we feel confident that we passed the Linux message to many thousands of people who otherwise had no real knowledge about the Linux revolution. Viva la revolución!

Resources

Chris DiBona (chris@dibona.com) is a computer security consultant and the Vice President of the Silicon Valley Linux User's Group (http://www.svlug.org/). He enjoys Linux, studying terrorism and population statistics. He also grooves on a good Pale Ale. His personal web page can be found at http://www.dibona.com/.

Cobalt Qube Microserver

Ralph Sims

Issue #54, October 1998

All about this compact web server hardware.

- Manufacturer: Cobalt Networks, Inc.
- URL: http://www.cobaltmicro.com/
- Price: $999 to $1449 US
- Reviewer: Ralph Sims

Solid geometry and writing magazine articles are two things with which I've never quite felt comfortable. My concern heightened when I was asked to write an article about Cobalt Qube for *Linux Journal*--but I figured what the hey, what's the worst that can happen?

Actually, my relationship with the Qube was not new. I was at ISPCON in Baltimore this past spring and was walking down an aisle daydreaming, when this blue cube with an intriguing green light-bar caught my eye. ISPCON can rather numb one's senses, so this diversion came at a good time.

The chap from Cobalt Networks, Inc. with the 7-inch cube-shaped box described it as a tightly integrated “microserver” that functions as a web server and provides other features such as web publishing, e-mail, private and public discussion groups, and file and text indexing and searching. “Well, cool, but I can do that with Linux” said I. “This runs Linux” said he. Now I was ready for the sales pitch.

The salesman told me this box contained a 64-bit super-scalar MIPS RISC processor running at 150MHz, 2.1GB UltraDMA IDE drive, 10Base-T Ethernet, 16MB RAM and a back-of-the-box LCD panel for quick setup.

The operating system for the Qube is Linux 2.0, ported to the MIPS CPU, complete with the Apache web server. It supports CGI, Perl 5.0 scripting, e-mail via SMTP, POP3 or IMAP4, Domain Name Service and the capability to drag and drop files from their Linux, Windows, Windows NT or Macintosh boxes using SMB or AppleShare.

The salesman also told me how easy it is to set up a browser, that user password control is available, that the Qube took either a static IP address or used DHCP, and that it is suitable for both Internet and Intranet applications. The clincher was the price—under $1000.

As an Internet Service Provider, I'm always looking for hardware which our current and potential customers might find useful. The Cobalt Qube 2700 Microserver could very well fill the need for a solid, easy-to-manage, easy-to-use web server that could be co-located in our data center or placed at a customer location and tied directly to either the Internet or the LAN.

For me, there was only one problem. Since the box is a cube, it is not suitable for most data centers, where rack space can be at a premium. “It would be great if it were available in a rack-mount profile”, I mentioned, and was told they'd look into it. A rack-mount unit is now available, taking up a whopping 1RU (1 rack-unit) or 1.75 inches of space. The Cobalt RaQ Microserver is an ISP's dream come true, with front-to-back ventilation, no-clearance mounting and a low-drain internal power supply. The fact that they actually *listen* to user input and act on it is impressive.

Ralph contemplates the Cobalt Qube

When I got back to the Puget Sound area, I showed the literature to a couple of folks and the general impression was “Neat. How can we use it?” An evaluation unit seemed to be in order and that's where Phil Hughes and the folks at *Linux Journal* came into play. While Linux has always provided a platform for the hobbyist and OS adventurer, until recently it has not enjoyed recognition as a player in the commercial world. The Cobalt Qube is doing its best to dissuade the naysayers. *LJ* got a Qube shipped to me and the fun began!

I had been running Linux since the “early days” and recently enshrined my 0.99pl4 boot disk. I knew a bit about the OS and its capabilities, so I had a preconceived notion or two about what to expect when I opened the box from Cobalt.

Documentation? I don't need no stinkin' documentation! I plugged in the external power supply and Ethernet cable and fired up the Qube. In anticipation of using it, I had already assigned an IP address from my Internet Domain Name Server with a reverse lookup to www.mousepotato.com. I was going to put it live on the Net!

Oops! Out of the box, the Cobalt Qube wants an address via DHCP (actually, the IP address can be addressed using DHCP or the LCD buttons, but remember, I didn't read the documentation). I unplugged the Ethernet and let the setup fail, then entered a static IP address, netmask and gateway. I plugged in the cable and the box dutifully continued its internal configuration and was up—a little over five minutes had elapsed.

I pointed a web browser to the IP and was presented with an opening screen. Having this device completely configurable over a network connection is greatly appealing, in particular for remote management. Linux boxes rarely lock up to the point where they can't be accessed remotely, and folks using the boxes aren't always at a local machine. The Qube doesn't allow for a keyboard or a monitor, so there is no alternative to remote access.

The Setup Wizard led me through host name and system administration functions. I set the time, an administrator (root) password and some basic access rights for a user, and was off and running.

Some of the user services the Cobalt Qube handles are e-mail, web page development, threaded discussions, information services and file sharing/data transfer. It allows for the forwarding of e-mail to an account at another location. In response to user input and needs, the folks at Cobalt have now integrated IMAP4, POP3 and SMTP into the box.

The Cobalt Qube understands the notion of “groups”. A number of users on the Cobalt Qube can be assigned to one group. Each group automatically gets an e-mail address, a public home page, a private home page and a discussion group accessible to the members of the group. For example, the marketing group can have a public site visible to the entire company while maintaining a private site and a private discussion group containing information on group projects.

Document management is also done through the browser using an “Infobase” which can contain text pages, graphics and multimedia files, and URLs, and is indexed on a scheduled basis. Indexing files and text is done using Glimpse in a maintenance event, and searches can be either single-word or phrase-based. Clicking on the returned file name opens that file in a browser.

I've never set up SAMBA before, so I was glad this task was already done. Dragging and dropping from a Windows box really made the day for my teenage boys, who are on their way to becoming good web developers. (I haven't quite gotten them to use Linux for everything.) Macintosh owners can use AppleShare. Everyone can use FTP.

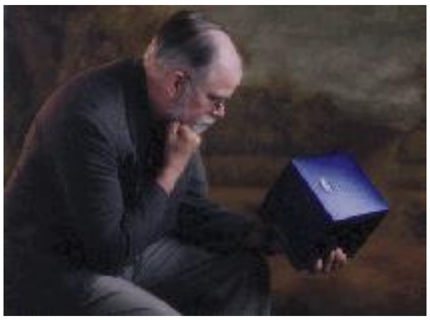
User access to the various services is handled on an individual or group level and is totally under the control of the system administrator.

Maintenance events are handled through the browser interface, and **cron** does the behind-the-curtain scheduling. Some features include persistent memory and CPU utilization, service status of web, e-mail, file services and a graphical representation of system problems and status. A browser-based backup and restore utility is available.

For those who wish to do things the old-fashioned way, TELNET access to the box is allowed. However, with the way things were laid out and the tight integration of services with CGI, I chose not to mess with operations from the command line. Before using this on a regular basis, I'd take some time to dissect the way things are put together.

For web publishing, there's a built-in server-side web page builder that's not too hard for the beginner. I had some basic problems at first (remember, I still haven't opened the book), but my oldest teenager came to the rescue. (I use him for my Windows tech support, too.) One can also use Netscape Composer or other web page development tools such as Frontpage or PageMill to create web sites and then transfer them to the Cobalt Qube using file sharing or FTP.

How well does the Cobalt Qube perform? I ran it through a rather basic test using a shell script on the LAN, firing off HTTP requests as fast as my Linux box could send them out. The server never missed a beat. However, it never seemed to get out of swapping to disk during the entire time I had it, and I'd suggest getting a bit more RAM to help out there. The folks at Cobalt state the performance is better than a 200MHz Pentium for random file access, thanks to the optimized disk drivers. The Cobalt Qube can serve around 100 web page requests a second (i.e., greater than 8 million hits a day). It can concurrently handle over 140,000 e-mails, 50,000 file transfers and over 250,000 web page requests a day—quite sufficient for most applications.



Performance of the Cobalt Qube was compared with competing products (200 MHz NT box, Whistle Interjet, Web Zerver, Cisco Microweb server, Twister). The performance is roughly equivalent to a 200MHz NT box and two to five times better than other microservers/thin servers.

Strong points are the Linux OS and the Apache web server. Cobalt's use of these products for the Qube shows support for the hard work done by the respective user and development communities and the commitment they have to giving back to those communities. The entire system is based on open, standards-based design. I like the idea of being able to tack on those tools and applications I find useful without being tied to someone else's idea of the way things should be. Cobalt has done an admirable job of anticipating the needs of web developers and system administrators.

The Cobalt Qube is solid. I wound up doing an ad hoc physical stress test by putting one in the trunk of my car for storage and forgetting about it. The roads over by Shelton, WA aren't the best around and slam-and-go traffic during Seattle's rush hour is aggressive enough on its own, but the box survived and the data remained intact. Even my cats pulling out the power cord numerous times hasn't caused a problem. Physically, the Qube is put together well, and Linux can recover from a lot of abuse. The engineers have now figured out a way to keep the power cord attached to the adapter block from coming out.

The Cobalt Qube comes in three flavors. A 2700D model contains web and e-mail servers and a full suite of development tools aimed at web and application developers. It has a 2.1GB disk and 16MB RAM. It is priced at $999. The 2700WG models are targeted at workgroups and have a full suite of services (web, e-mail, file sharing, discussion groups, text indexing and retrieving). The two models have 2.1GB/16MB and 6.4GB/32MB disk and memory and are priced at $1249 and $1449, respectively.

I'd recommend the Cobalt Qube for web developers who are putting together sites for their customers—creating a “web site in a box”, as well as for companies and organizations that need to share data internally on their LAN. The Cobalt Qube is the only product in its category that allows custom CGI/Perl development, an essential for today's interactive web sites. The user interface is uncomplicated and the ability to manage the system remotely gives it high marks. It should make an administrator's life much easier.

Quite a bit of information is available at Cobalt's web site, http://www.cobaltmicro.com/, and it's worth reading to see how Linux is working its way into the mainstream of corporate information services.

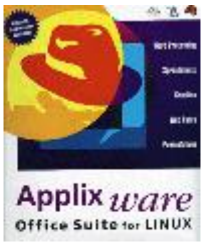
Ralph Sims is Vice President of Northwest Nexus, Inc., a Regional Internet Service Provider in the Pacific Northwest. He started using Linux at his company in 1993 on a 386/25 that provided free e-mail and news access under the Waffle BBS frontend. He provides a mirror of the Linux kernels and Slackware release (his favorite of all the distributions) at ftp.linux.locus.halcyon.com. He lives in Shelton, WA with his two teenage sons and a handful of cats. He can be reached via e-mail at ralphs@nwnexus.net.

Applixware and StarOffice

Fred Butzen

Issue #54, October 1998

A detailed comparison of the two office packages, their installations and ease of use.



- Product: Applixware
- Manufacturer: Applix, Inc.
- E-mail: applixinfo@applix.com
- URL: http://www.applix.com/
- Price: $49.95 US with Red Hat 5.1

- Product: StarOffice
- Manufacturer: Star Division, GmbH
- E-mail: info@stardivision.com
- URL: http://www.stardivision.com/
- Price: $199 US with Caldera OpenLinux 2.1
- Reviewer: Fred Butzen

Linux has proved itself to be enormously successful among computer cognoscenti—those who make software and run networks. However, it has proved less successful among ordinary computer users—those who buy software and pay for networks. Whenever I describe Linux to ordinary users, I inevitably receive two reactions:

1. They are delighted to hear there is another operating system available for their PC, particularly one as powerful and as inexpensive as Linux.
2. They wonder what they “can do” with Linux. In particular, they wonder which Microsoft Office applications they can run under Linux. When I answer “None,” the light fades from their eyes as their interest dies.

As much as Linux enthusiasts may deplore this attitude, it is certainly natural. Ordinary users, after all, care little about the technical features of an operating system, and even less about the politics of it. What matters to them is whether they'll be able to use the computer to do their work; and whether we like it or not, the Microsoft Office suite of applications has been established in most workplaces as the standard suite of tools by which work gets done.

Fortunately, two packages will let me change my “No” to a tentative “Yes”. Applixware and StarOffice are two robust, integrated office packages that offer reasonable compatibility with the Microsoft Office suite of tools.

In this review, I'll discuss and give my impressions of each package.

Applixware

Applixware is manufactured by Applix, Inc., of Westboro, Massachusetts. I used release 4.3, in the version designed for use with Red Hat Linux 5.0. The package comes with the base release of Red Hat 5.0; however, all executables are statically linked and should work with any release of Linux using kernel 2.0 or higher.

Applixware includes the following modules:

- Applix Words, a word processor
- Applix HTML Author
- Applix Graphics, a tool for creating and editing graphical presentations (slide shows and the like)
- Applix Spreadsheets
- Applix OpenMail, a mailer
- Applix Presents, a tool for creating slide presentations
- Filter packs to convert data from and into a variety of popular formats

The utilities use a common object format, so that objects built with one tool can be integrated into another. For example, a spreadsheet built with Applix

Spreadsheets can be integrated into an Applix Words document, or sent as a mail message via Applix OpenMail.

An optional utility, Applix Builder, gives a user lower-level access to the objects built with any of the utilities. The user can integrate these objects with custom code to construct new customized applications.

Finally, the package comes with a printed manual that introduces Applixware and gives the basics of how to work with the package as a whole.

StarOffice

StarOffice is an office package that is manufactured by Star Division GmbH, of Hamburg, Germany. I used release 3.4, in the version bundled with Caldera's OpenLinux. This edition of StarOffice is usable on most Linux systems with kernel 2.0 or later and libc 5.4.4 or later.

StarOffice includes the following modules:

- StarCalc (**scalc3**), a spreadsheet
- StarChart (**schart3**), a program for creating charts and graphs
- StarDraw (**sdraw3**), a tool for assembling slides and presentations
- StarImage (**simage3**), a tool for working with images
- StarMath (**smath3**), a mathematics tool
- StarWriter (**swriter3**), a word processor

Also included are two daemons:

- **svdaemon** manages on-line help
- **svportmap** manages communication among the modules

StarOffice does not come with a printed manual, at least in the edition I tested. Given that Star Division GmbH is a German company, the native language of the package is German; as a result, the documentation and notes are worded a bit oddly at times.

Applixware Installation

Installation of Applixware requires the user to log in as the superuser. Installation for Applixware is managed by **rpm**, the Red Hat Package Manager, a copy of which is included on the Applixware CD-ROM. A shell script invokes **rpm**, queries the user via a crude graphical interface for options (e.g., the directory into which the package should be installed and alternate languages for documentation), then installs the selected Applixware features. The default installation, including optional clip art and English-language documentation, consumes about 112MB of disk space.

Applixware's various applications work through a single master program, called **applix**. Applixware creates a symbolic link to **applix** in /bin at installation. Because **applix** manages access to shared libraries and other utilities, a user does not need to modify her environment to use Applixware.

The license for this version of Applixware limits it to a single user.

In brief, installation is simple and straightforward.

StarOffice Installation

Installation of StarOffice requires the user to log in as the superuser, then use **rpm** to install packages of executables. A copy of **rpm** is included with the release, in case your release of Linux does not include it. Unlike Applixware, in which installation is managed by a shell script, you must invoke **rpm** by hand for each part of StarOffice you wish to install.

Installation goes quite smoothly. The default installation, including English-language documentation, consumes about 75MB of disk space.

The license for StarOffice allows the package to be used by only one user. That user must run a set-up program to configure the package for her. She must then add a script to her account's profile to set the environment properly for StarOffice.

Installation and setup are relatively simple. Unfortunately, the need to patch the user's environment is not documented. A naive user will find StarOffice does not work after setup and will have no way to figure out how to fix the problem.

Test Environment

I tested both packages on a home-brew PC built around a 100-MHz i80486 DX4 processor. The machine has 16 megabytes of RAM and a 2.2-GB SCSI disk plugged into an Adaptec 1542 controller. The machine runs Slackware release 3.2, which uses kernel 2.0.19 and libc release 5.4.17.

Both packages require a lot of horsepower. The test system was able to run both reasonably well. However, running either package on a 486 is not something I would recommend if you are at all impatient.

Criteria for Testing

To exhaustively test each module in two complex packages is impossible in a reasonable period of time. I decided, therefore, to concentrate on the following criteria for each package:

- *Functionality:* Does each module provide at least the minimum needed to perform a given task?
- *Communication:* Do the modules work well with each other?
- *Portability:* Do the modules import and export files correctly? I was particularly interested in their ability to import and export files to Microsoft Office.
- *Robustness:* Could I find any obvious bugs?
- *Aesthetics:* Which package's "look and feel" did I prefer? This is, of course, a subjective judgment.

Applixware Results

As mentioned earlier, Applixware works through a single master application, which invokes all of the other applications.

First, here are my global impressions. A user who is familiar with Microsoft Office will find herself at home in Applixware. The biggest difference is that toolbars and buttons cannot be rearranged dynamically—what you see is what you get. Likewise, it does not come with as rich a set of fonts and templates as Office. The contents and ordering of menus, including right-button menus, closely resemble their Microsoft analogues.

A few minor differences exist. For example, you must click a menu's button before the menu drops down, rather than just sweeping the mouse pointer over the button. I, for one, find this to be an improvement rather than a problem; you may disagree.

As with the Microsoft Office applications, each object manipulated by Applixware is linked with the utility that generated it. For example, if you drag clip art into an Applix Words document and then double-click over it, Applix invokes Applix Graphics, which displays the graphic and allows you to edit it.

Finally, Applix comes with a nicely written set of documentation, including context-sensitive help screens.

Applix Words is a full-featured word processor. Figure 1 shows the Applix Words screen with some text dropped into the document. (Images in this article were made using **xv**.)

Figure 1. Applix Words Screen

My preliminary test showed it to be robust. I found I could export documents to Microsoft Word and import documents from it without error. Unlike Microsoft Word, the user must click the **Import** button and give the document's type, rather than clicking the **Open** button and depending upon the application to sense the file's type; however, I found this to be only a minor difficulty.

An anecdote may illustrate the application's robustness: while I was working on this review, I was also doing the index for *The Linux Network*. The publisher insisted on receiving it in Microsoft Word 6.0 format. I prepared a raw form of the index in text under Linux using Emacs, because Emacs is a much more powerful editor than any available in a GUI environment. My intention was to convert the file to Word format just before I shipped it. However, the weekend before I was to deliver the index, the hard disk on my Windows 95 machine died. I did not have time to install a new disk and reconfigure it. In desperation, I used Applix Word to convert the index to Word format and shipped it to the publisher; the file was accepted without a murmur.

Applix HTML Author is a simple, straightforward tool for preparing web pages. I found it useful for editing existing web pages, less so when preparing a web page *de novo*. One drawback is that HTML data types are presented in the menu by name rather than through descriptive text.

On the whole, a user who is not already familiar with HTML will not find this tool especially helpful.

Applix Graphics is a tool for drawing and assembling graphics. Anyone who has used similar tools, such as Frame, Microsoft Word or **xfig**, will find its features and interface familiar.

Applix Graphics serves as the graphics “lightboard” for Applix Presents (described below) and other Applix applications. Created images can be exported to most common graphical formats and can be pasted into other Applix documents. In this, Applixware has a definite advantage over Microsoft Office. Office's applications each have their own graphics engine, and the output of each is by no means totally compatible with the other Office applications. (As I found once when I tried to export a PowerPoint graphic to Word 6.0, only to find that all gray-scale elements had vanished.)

Applix Spreadsheets is a spreadsheet with the usual features. I am no spreadsheet maven, so I will pass over it by saying that it appears to do what one would expect from a spreadsheet.

Applix Presents: This tool lets the user build graphical presentations such as slide shows, sets of overhead displays or handouts. It is analogous to Microsoft PowerPoint. As I mentioned earlier, this tool uses Applix Graphics as its “lightboard”, so if you've become familiar with that tool, you should not have much trouble mastering Presents.

This tool comes with a nicely designed set of templates.

Users who are used to PowerPoint will find this tool departs from the Microsoft “look and feel” somewhat more than the other Applix utilities. However, most of the useful PowerPoint features are present, including outline format, notes format and the ability to print in a variety of single-slide and multi-slide arrangements.

Applix Mail is a mailer which contains the usual features, such as those found with Netscape Communicator or similar GUI mailers. One nice feature not seen in Netscape Communicator mailers is that messages with attached files built by Microsoft applications can be exported immediately to the appropriate Applix utility and viewed. You don't have to save it to a file, then export it to a Windows box before you can view it. Not every Microsoft format is recognized; in particular, a filter for Word 7.0 is not included with the package reviewed. However, I nonetheless found it to be quite useful.

StarOffice Results

As I mentioned earlier, I found StarOffice more difficult to install than Applixware, although an experienced Linux user should not find installation of either package particularly difficult.

Unlike Applixware, in which each application is invoked through a master control-panel utility, StarOffice consists of individually invoked applications that communicate via a special daemon. Unfortunately, I could not get either the inter-application daemon or the help daemon to work under Slackware. In all likelihood, this was due to my system's lacking some special configuration; still, without the help daemon working, I had no way to read the documentation to discover the cause of my problem.

First, here are my global impressions. The StarOffice package is quite sophisticated. Its look-and-feel is slicker than that of Applixware—clearly some very talented designers and programmers paid a great deal of attention to the details of this package. While its interface will be easily used by someone who has worked with the Microsoft Office tools, it is, in my opinion, more flexible and easier to use than its Microsoft counterpart. There's something very German about StarOffice, in the same way there's something very German about a Mercedes or a BMW—a combination of impeccable engineering and a design that is solid, rich and intelligent.

As with the Microsoft tools and unlike Applixware, StarOffice implements each element of the interface as a separate object. This lets the user assemble and arrange the elements of the interface to best suit her preferences.

This look and feel does come at a price, however; StarOffice runs considerably slower on trailing-edge hardware, such as the system I used for testing, than Applixware does. This is serious software that requires serious hardware. The subtlety of the interface, too, requires at least a 17-inch monitor to be used to its best advantage.

StarCalc is the spreadsheet application included with StarOffice. As I mentioned earlier, I seldom use spreadsheets; however, as best I can tell, StarCalc offers the features that one would expect from a well-designed spreadsheet.

StarChart builds charts and graphs. The user can either enter data by hand or use files built with any of a variety of word processors, text processors or spreadsheets. StarOffice allows the user to build the standard variety of charts: pie, bar, line, scatter or area. Images or graphics in any of a variety of formats can be pasted in. Graphs can be exported to other StarOffice applications, or to a variety of other tools.

StarDraw is a tool for assembling images. With StarDraw, you can assemble images and graphics plus text into presentations of one or more slides. This is the StarOffice analogue to Microsoft PowerPoint.

StarImage is a tool for working with images. It comes with the usual features for assembling and manipulating images. However, I don't think it is a tool that will make anyone forget **xv**.

StarMath is a tool that enables building and testing of mathematical formulas. You can import formulas from other tools, use a point-and-click interface to select from a rich set of mathematical elements in order to modify a formula, and export the results to a variety of other tools, including StarCalc.

Figure 2. StarWriter's Main Window

StarWriter is a full-featured word processor. Figure 2 shows StarWriter's main window. The StarWriter interface will be familiar to those who have worked with Microsoft Word. Folded into StarWriter are a number of features usually seen in separate tools, in particular tools for displaying and building documents in HTML. What a novel idea: a tool for working with documents should work intelligently with a variety of documents, regardless of format.

I should state here that I have spent thousands of hours working with text processors, starting with **troff** and working through Word, Frame and other WYSIWYG tools. Over that time I've become rather jaded; I've found that slickness in a tool usually impedes my work rather than helps it. But this jaded user quickly fell in love with StarWriter; it is now my word processor of choice. Now, if I could just convince the boss to let me use it at work.

Conclusion

I was delighted with both packages. I found them both to be good, robust office tools. I warmly recommend both Applixware and StarOffice to users who need to run office applications under Linux.

Which package you choose will depend on a number of factors:

- Applixware's installation is easier than StarOffice's. Applixware does not require a user to modify her environment before using the package.
- Applixware runs more briskly than StarOffice, at least on the trailing-edge hardware I used.
- The edition of Applixware I reviewed includes a printed introductory manual, which I found helpful. StarOffice requires users to gain access to the on-line documentation, and doing so is by no means easy.
- StarOffice's interface more closely resembles Microsoft Office than does Applixware's. StarOffice's buttons, switches and menus will appear familiar to users who have worked with Microsoft Office.
- StarOffice's interface is configurable: buttons can be added to the toolbars, removed from them or repositioned. Applixware has a static interface: what you see is what you get and that's it.
- Each package offers some applications not included with the other. For example, Applixware includes an HTML-builder, whereas StarOffice includes a mathematics package.

These differences notwithstanding, each package offers excellent value for your money, especially compared to its Microsoft counterpart.

If you are considering an office package for Linux, I strongly suggest that you try out both packages and select the one that best suits your needs and tastes.

In my opinion, both Applixware and StarOffice offer the ordinary computer user a reasonable alternative to Microsoft Office, and offer other features as well—not to mention giving the user access to the power of Linux.

Fred Butzen is a technical writer and programmer who lives in Chicago. He is the principal author of the manual for the Coherent operating system, and is co-author of The Linux Database (MIS:Press, 1997) and *The Linux Network* (MIS:Press, 1998). He can be contacted at fred@lepanto.com.

Archive Index Issue Table of Contents

Advanced search

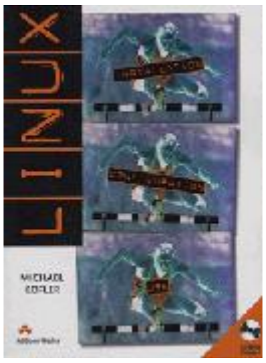
Advanced search

Linux: Installation, Configuration, Use

Michael Scott Shappe

Issue #54, October 1998

What new users need is a friendly guide to show them the ropes.



- Author: Michael Kofler
- Publisher: Addison-Wesley Longman, Harlow, Essex, England
- E-mail: kofler@ping.at
- URL: http://www.addison-wesley.de/Service/Kofler/home.html
- Price: $34.95 US, $47.95 CAN (includes CD-ROM)
- ISBN: 0-201-17809-5
- Reviewer: Michael Scott Shappe

Linux can be a scary environment to the uninitiated. Users familiar with only GUIs can have a hard time adjusting to the plethora of typed commands. Those familiar primarily with control panels suddenly have to edit /etc files with strange syntaxes that vary from file to file. Even DOS users, used to typing commands and editing configuration files, can find the sheer power Linux places in even the average user's hands a bit daunting.

What new users need, then, is a friendly guide to show them the ropes. They need a tome that gives a broad overview of what they can do with this new operating system they've been hearing so much about, but that doesn't go into such depth as to scare them away. *LINUX: Installation, Configuration, Use* (first published in Germany as *LINUX: Installation, Konfiguration, Anwendung*) is a very good starting point.

Making the Switch

This book is aimed primarily at experienced users of other environments, such as DOS or Windows—people who know how to turn their computer on and off, who know how to get what they want out of what they already have and a little more besides, and are now ready to try something new.

The book has a surprising breadth of topics and gives what I think is just the right amount of depth for its purpose: a general overview of how to put a Linux system into practical use. In every area, it gives the user enough information to get him going, and then a little more to make it clear that any limitations are with the scope of the book and not the capabilities of the environment. It then points the reader to resources needed to go further.

The book begins, appropriately enough, with a brief description of Linux's history, along with an overview of what's available for Linux, and what sorts of things—both technologically and philosophically—make Linux unique. From there it moves immediately into the realm of the practical, walking the user through the first installation of Linux on his system.

The Installation chapter covers several different scenarios, but focuses primarily on a single distribution: Red Hat 4.1. This distribution is included on a CD with the book—a common enough occurrence—so the book tends to fall back on Red Hat when giving specific examples. Other distributions—notably Debian, Slackware, Caldera and S.u.S.E.—are mentioned, and significant differences are dealt with.

Installation taken care of, the author moves right along to a whirlwind tour of UNIX in general and Linux in particular. This section goes into no particular depth, but rather gives a rapid-fire overview of the basic utilities and programs a user should become familiar with, such as **more**, **emacs**, **vi** and X. Before diving further into the nitty-gritty, the author makes it clear where a user can find all the extensive on-line documentation—both from the included CD and also information available on the Net.

Making it Work

The section on configuration opens with a discussion of file management under Linux. Here again, the assumption is that the user is at least somewhat familiar with the concepts behind other command-line interfaces, such as DOS, although I don't believe a complete newbie would be lost. The section goes into how Linux structures file systems, how permissions work, the basics of administering users and groups and coping with removable media.

Following this is a peek at the actual innards of file systems—not in much detail, but enough to whet the appetite of a curious novice. Process administration comes next, followed by a discussion of library-related issues, including some troubleshooting tips for shared libraries. Finally, the **init** daemon and its importance are described in some detail before moving on to the topic of configuration.

The basic configuration and administration chapter starts with simple things—keyboard configuration; configuration of BASH, **less** and Emacs; setting the time, that sort of thing. It then walks the user through setting up new accounts, and moves on to a description of how to administer file systems. Next comes a discussion of LPD, followed by network configuration. The chapter concludes with a detailed section on tailoring the boot process, and a walk-through of how to recompile the Linux kernel.

The section on configuration concludes with a description of the X Window System, focusing on XFree86 3.1.2 (the version on the disk). All of the basic configuration issues are discussed here, including the importance of getting the monitor settings right so that you don't accidentally fry your screen. An overview of window managers is provided, covering several flavors of FVWM, TWM and OLWM. The section concludes with a quick overview of X Resources.

Making it Useful

The third section of the book focuses on teaching someone how to actually *use* the system. An entire chapter is devoted to the bash shell and the basics of writing bash scripts. Following this is a 50-page command reference, covering just about every executable that comes with the standard Red Hat distribution.

Having established the environment one executes commands in and the commands one can execute, subsequent chapters cover the use of specific programs. First comes a chapter on tools and utilities, including Midnight Commander, various PostScript tools, **xv** and **xgrab**.

After this, an entire chapter is devoted to Emacs, starting with the basics of finding your way around the editor and going right up through advanced issues like macros. Next comes a chapter on LaTeX2E, again starting with the basics but ultimately at least touching on more advanced features. Between these two chapters, the user is left with enough information to begin word processing—albeit in a non-WYSIWYG environment—under Linux.

This section finishes with an extremely useful overview of how to get a Linux box up and running on the network. In addition to holding the user's hand through the dreadful process of setting up a PPP script and other such details, the book provides an overview of web, FTP, TELNET, e-mail and news clients available for Linux. The section on e-mail even covers sendmail configuration, and includes instructions for setting up a machine to use **sendmail** and **popclient** to send and retrieve mail from a remote system using PPP.

Going Further

Finally, the book spends about 100 pages showing the user several different environments in which he can write programs—the real power of the Linux environment. Rather than delving into the mysteries of C and C++, this section focuses on more “mundane” environments—bash, Tcl/Tk and Emacs Lisp. In all three cases, the emphasis is more on giving the user enough information to extend his day-to-day working environment, rather than on extensive programming.

The book winds up with a set of appendices that cover the vagaries of various distributions and deal with updates since the original, German edition of the book was published.

A Tough Job (But Someone Has to Do It)

This kind of book cannot be easy to write and must have been even harder to translate. That said, I think Mr. Kofler and whoever translated the book from its original German have done a fairly good job. I've been administering and programming UNIX systems of various stripes for years and have been running Linux both at home and at work on a daily basis for quite some time. I still found Mr. Kofler's book immediately useful, enabling me to easily tackle a couple of issues I'd been putting off, waiting for a block of learning time.

I have, I think, only two complaints about the book. The first is that it's already slightly out of date: Red Hat 4.1 is over a year old. On the other hand, Red Hat 4.1 is also a solid release—it's the release I was already running on my laptop, and I've had very few problems with it. However, some things made the age of this release noticeable. For example, the section on setting up an automated script to dial in with PPP, send queued mail, retrieve mail from a POP box and disconnect, was very useful. But it relied on an older POP program—popclient—rather than the newer and more broadly useful (not to mention supported) **fetchmail**.

The second complaint is that the translation to English is not quite as good as it could be. There's nothing embarrassing about the translation; it's just not quite fluid sometimes. Many readers won't even notice, but as a copy editor, I do.

As an example, on page 141, preparing to describe how to break up a Linux system into multiple partitions, I found this sentence: “The creation of additional Linux partitions is a far-reaching intervention in the Linux system.” Now, there's nothing *wrong* with that sentence, from either a syntactic or a semantic point of view, but the choice of words (“far-reaching intervention”) seems a little strange. I found things like this a bit distracting, but again, that may be because I look for them as a matter of habit.

Overall, I found this book extremely useful with a very positive attitude and a wide range of topics covered. As both a step-by-step guide and a reference, it is a good place for an aspiring Linux user to start.

Michael Scott Shappe is a somewhat frazzled software engineer for AetherWorks Corporation, a start-up in Saint Paul, Minnesota. When he's not writing reviews or copy editing for *Linux Journal*, he's reading—and attempting to write—fiction, or attending Society for Creative Anachronism events. He can be reached at Mikey@Hundred-Acre-Wood.com, and his web page is at http://www.14850.com/web/mikey/.

Archive Index Issue Table of Contents

Advanced search

Embperl: Modern Templates

Reuven M. Lerner

Issue #54, October 1998

Mr. Lerner introduces us to a template system for Perl: what it is, how it works and how to use it.

Earlier this year, I described mod_perl, a module for the Apache web server that embeds a full version of Perl inside Apache. Not only does this allow you to write CGI-style programs that overcome CGI's bottleneck problems, but it also gives you access to Apache's innards, letting you configure your server in many new ways. A number of developers have begun to take advantage of this flexibility, configuring Apache in new and clever ways.

One such clever idea is Embperl, written by Gerald Richter (richter@dev.ecos.de). Embperl allows you to create hybrid pages of HTML and Perl. As we have seen in several previous columns, templates allow designers and programmers to modify their respective parts of a web site without getting in each other's way. If the programmer wants to modify the logic, he or she can do so by modifying the Perl parts of a template. By the same token, designers can modify the look and feel of a page without having to ask the programmer to change a few **print** statements in a CGI program.

Embperl is but one of several template systems available for mod_perl. Another contender for this role is ePerl, about which I have read quite a bit, but haven't yet had a chance to try. Another solution, which uses Perl but doesn't depend on mod_perl or Apache, is Text::Template, a module I have used in previous columns when discussing templates. Finally, PHP is an embedded scripting language that resembles C and Perl in many ways, and is designed to be interspersed with HTML inside of documents. To find more information about all of these, including URLs, see Resources.

How does Embperl work?

Before we can use Embperl, it's important to understand how HTTP requests and responses are formed, and how a web server performs its job. When you click on a web page link, your browser connects to the host name in the URL and sends a short request to the server. The request consists of a verb (typically GET or POST), the name of the document being requested, and the version of HTTP that the browser supports. For example, to request the root document from a web server, a browser will typically send

```
GET / HTTP/1.0
```

to the server. It is the server's responsibility to handle the request, responding with an error message or a document. Depending on which version of HTTP the browser is running, the server might return multiple documents in the same response, demand some sort of user authentication before continuing, or redirect the user's browser to a different URL.

In many cases, though, the server will not return a document at all. Instead, it will run a program, returning the program's output, rather than its contents. This is how CGI programs work: the server is configured such that all files in a certain directory are treated as programs, rather than documents to be retrieved verbatim. (Indeed, security concerns arise when users can retrieve programs' contents, rather than seeing their output.) As far as the browser is concerned, it requested a document and received one in response. The magic happens on the server side, where the program is executed and produces its output.

A price is paid for CGI programs, above and beyond their execution times: because web servers fork a separate process for each CGI program, and Perl (and other popular scripting languages) can have a long start-up time, it often takes longer for the program to get started than for it to actually run.

For this reason, each web server has developed its own native API that allows programs to bind more closely to the server's internal code than would be possible with CGI. Netscape's NSAPI and Microsoft's ISAPI are two examples of such proprietary systems, and Apache's mod_perl is an example of how similar functionality can be given to Perl programmers. With mod_perl installed in your server, operations speed up tremendously, because the server compiles the program once, rather than each time it is run. In addition, because the program never requires creating a separate process, the overhead associated with executing such programs is relatively low.

Mod_perl is perhaps best known for allowing programmers to write very fast CGI-like programs. However, since Apache's internals are available via mod_perl, it is possible to write Perl programs that change one or more steps in Apache's processing of outgoing documents. These can range from the mundane to the fancy; in Embperl's case, we are setting a special PerlHandler for particular documents. In the Apache world, a “handler” is a program that does something special with the files in a directory before returning them to the HTTP client. You can think of a handler as a middleman between Apache and the file; the handler grabs the file and modifies it as necessary, handing the finished product to Apache. Apache then takes this finished product and returns it to the user's browser in the HTTP response.

Installing Embperl

Note that installing Embperl can be a bit tricky. The documentation is generally good and describes all of the steps necessary to install it on your own computer. I have installed it several times and found that each time required several tries before I managed to follow the directions correctly. I will describe the procedure here in some detail, but you might want to look at the FAQ file that comes with Embperl for more information. (Many of the following instructions are based on that FAQ.)

Before you begin, you should install or update the latest versions of several packages: LWP (the library for Web client programming), HTML::HeadParser (used for parsing HTML document heads), CGI.pm (the super-module that handles everything having to do with CGI), and MIME::Base64 (which handles the encoding information to and from Base64, which is used in the MIME standard). All of these are available from CPAN (see Resources).

Both mod_perl and Apache must be recompiled in order to get Embperl running. It is possible for Embperl to run as an external CGI program, rather than from within mod_perl, but you will then lose the speed benefits of mod_perl. I strongly suggest going the mod_perl route, unless you are using a web server other than Apache, or if you would rather not recompile things just now.

For starters, then, you will need the source for Apache (from http://www.apache.org/), mod_perl (from CPAN, at http://www.perl.com/CPAN/) and Embperl (also from CPAN, as HTML-Embperl-1.0.0). On my machine, these packages were named as follows:

```
HTML-Embperl-1.0.0.tar.gz
apache_1.3.0.tar.gz
mod_perl-1.12.tar.gz
```

I am sure that newer versions of these programs will be available by the time you read this article. However, you should be able to follow this discussion by updating the version numbers as appropriate.

First, unpack all of the files using the command:

```
for file in `ls *gz`; do tar -zxvf $file; done
```

Before you can start to compile the components, you will have to set some of the configurations and modify Makefiles. First of all, go into the mod_perl directory and edit src/modules/perl/Makefile:

```
/downloads/mod_perl-1.12/src/modules/perl/Makefile
```

You will have to make three changes to this file. First, add the HTML::Embperl to the definition of STATIC_EXTS that will be grabbed by the mod_perl configuration system. That is, edit the line (line 98, in mod_perl-1.12):

```
#STATIC_EXTS = Apache Apache::Constants
```

and change it to:

```
#STATIC_EXTS = Apache Apache::Constants HTML::Embperl
```

Next, look for the line that begins with **OBJS=** (line 131 in mod_perl-1.12). Just before that line, define the variable **EPDIR** so that it points to your Embperl build directory. For instance, assuming that we are building Embperl in /downloads/HTML-Embperl-1.0.0, we will set it to:

```
EPDIR=/downloads/HTML-Embperl-1.0.0
```

We will now modify the OBJS variable such that it creates the object files for Embperl as well as mod_perl:

```
OBJS=$(PERLSRC:.c=.o) $(EPDIR)/Embperl.o \
$(EPDIR)/epmain.o $(EPDIR)/epio.o \
$(EPDIR)/epeval.o $(EPDIR)/epcmd.o \
$(EPDIR)/epchar.o $(EPDIR)/eputil.o
```

Don't forget to put backslashes at the end of each continued line, so that **make** doesn't think the second and third lines should stand on their own.

The hardest part is over. All we have to do now is configure and compile the various components. Make sure to do them in the right order, though, or things might not work correctly.

First, enter the mod_perl directory and create the Makefile using the standard Perl command **perl Makefile.PL**.

If you want some or all of mod_perl's capabilities, now is the time to specify that. I tend to activate all of them (except for two that need explicit activation), so I enter **perl Makefile.PL EVERYTHING=1**. This will begin the mod_perl configuration process. You will be asked if you want to use the Apache source code in the parallel directory, and then if you want mod_perl to build **httpd** for you. Answer “yes” to both questions.

When the configuration script has finished running, go into the Embperl directory (/downloads/HTML-Embperl-1.0.0) and configure the module using the same command:

```
perl Makefile.PL
```

Once again, the system will perform a variety of configurations. You will be asked if you want Embperl to support Apache, and then if it should use the Apache source code in the parallel directory. Again, answer “yes” to both questions. Finally, you will be asked for a path name to the copy of httpd that will be used in testing. Check that the default is correct, and correct it if necessary.

Now we can actually create Embperl by typing **make** in its directory. After the compilation is complete, switch back to the mod_perl directory and create mod_perl and Apache by typing **make**.

Congratulations. You should now have working copies of Apache, mod_perl and Embperl. At this point, we could run **make install** in each of the three directories to install the software, or we can test Embperl. If you are interested in testing your Embperl compilation without installing it, I suggest that you read the FAQ. The directions are not that difficult to follow, but they are more complex than I can describe in the space provided here.

Configuring Apache to Allow Embperl

Now that Embperl is part of your copy of Apache, what can you do with it? Not much at this point, since we have not yet defined the handler for our Embperl files. Now we will have to modify Apache's configuration files, which might be in a number of possible places. When I installed Apache, I accepted the default installation locations (under /usr/local/apache), and my path names will reflect that.

In order to get Embperl working, we will need to modify two of Apache's configuration files. (Each of the three files can actually contain any of the configuration directives, but certain items are traditionally put in certain files.) I told the server to redirect URLs beginning with /embperl to /usr/local/apache/share/embperl by adding the following lines to the srm.conf file:

```
Alias /embperl /usr/local/apache/share/embperl
```

Next, I told Apache to install Embperl as the handler for that directory. As I mentioned above, this means that HTML::Embperl will be called each time Apache is asked to retrieve a document from /embperl. The file will be read from disk, handled by Embperl, and finally given to Apache, which returns it to the user's computer. I added the following to the access.conf file:

```
<Location /embperl>
 SetHandler perl-script
 PerlHandler HTML::Embperl
 Options ExecCGI
 </Location>
```

Once we have installed these changes, we restart Apache with:

```
/usr/local/apache/sbin/apachectl restart
```

Embperl Syntax

Embperl files look just like HTML files, with a minor difference: square brackets signify special sections of code, which are interpreted separately. In other words, you can put stock HTML files in an Embperl directory, although I would tend to advise against doing so, because of the additional overhead involved. Why force Embperl to look at a file unnecessarily? For that reason, some sites have decided to use a special suffix—.htmpl, perhaps—and then to configure Apache so that all files with that suffix, regardless of directory, are interpreted. That allows HTML files to be mixed in with their Embperl counterparts.

The following file, when retrieved from within a directory defined for Embperl, will print the current time:

```
<HTML>
<Head><Title>Current time</Title></Head>
<Body>
<P>This is Embperl</P>
<!-- Below are the square brackets -->
<P>[+ localtime(time) +]</P>
</Body>
</HTML>
```

Retrieving this file from an Embperl directory will produce the same output as the following short CGI program:

```
#!/usr/bin/perl -w
 use strict;
 use diagnostics;
 use CGI;
 # Create a new instance of CGI
 my $query = new CGI;
 # Send a MIME header
 print $query->header("text/html");
 # Send the HTML
 print $query->start_html(-title =>
        "This is Embperl");
 print scalar localtime(time);
 print $query->end_html;
```

However, Embperl has several advantages over a CGI program. For one, running it under mod_perl gives it a distinct speed advantage. Of course, we could modify our CGI program and/or Apache configuration so that the program would run under Apache::Registry, the mod_perl module that handles CGI-like programs.

The biggest advantage, though, is the clean separation between static and dynamic content. No longer does the programmer become the bottleneck, slowing down design and content changes—now the site's designers and editors can modify the HTML, so long as they stay away from the Perl inside square brackets. There will obviously be times when the embedded Perl code will affect the design, and the programmer can be included in such cases. But for the most part, such a separation allows everyone to do what they do best.

We have already seen one form of the Embperl brackets, namely **[+** and **+]**. Anything in square-plus brackets is evaluated as Perl code, with the results inserted into the HTML document and passed to the browser. Remember that the result of evaluating a Perl variable or string is the value of that variable or string. It's very common for square-plus brackets to contain a single variable name, whose contents are inserted into the document at the indicated point. Don't use the **print** function to insert things into the Embperl document, because **print** sends output to STDOUT, and then returns a result indicating whether it was successful. Each set of square-plus brackets can contain as much or as little Perl as you might like, although most Embperl programmers seem to prefer keeping the lines short.

Output from square-plus brackets is placed directly into the file without any additional formatting. If you want something to be in paragraph tags, boldface, italics or a different font, it is your responsibility to make sure that happens. Most often, you will want to surround the square-plus brackets with the appropriate HTML tags, so that the resulting output will be correctly formatted. That is, you could make a variable's value italic by saying

```
[+ "<i>$variable</i>" +]
```

but, for the sake of maintenance and separating static and dynamic content, it's better to say:

```
<i>[+ $variable +]</i>
```

What if you don't want the results of your code to be inserted into the document? You could end each set of Perl expressions with the empty string, as in:

```
[+ $counter++; &get_user_info($id); "" +]
```

which will insert the empty string into the document, since it was the last element to be evaluated. But a better solution would be square-minus brackets ([- and -]), which do that for you automatically. For example:

```
<HTML>
<Head><Title>Current time</Title></Head>
<Body>
<P>This is Embperl</P>
<!-- Square-plus brackets -->
<P>[+ localtime(time) +]</P>
<!-- Square-minus brackets -->
<P>[- localtime(time) -]</P>
</Body>
</HTML>
```

Output from the above Embperl will look the same as the one without square-minus brackets, since the output from operations performed in square-minus brackets aren't inserted into the HTML. This is useful when making variable assignments, as well as when importing other Perl modules. For example:

```
<HTML>
<Head><Title>Print user information</Title></Head>
[- $user = $ENV{"REMOTE_USER"}; -]
<Body>
<P>This is Embperl</P>
<P>[+ &print_user_profile($user) +]</P>
</Body>
</HTML>
```

Keeping Variables' Values

As you can see, variable assignments are kept across square brackets, meaning that you can assign a variable in one block and refer to it later. Variables are global by default, but you can use Perl's “my” convention to create temporary variables, which go out of scope at the end of the block.

One of the nice things about mod_perl is that it compiles programs once, caching them for future invocations. Not only do you save the overhead of forking a new process, but the program runs much faster since it only needs to be interpreted. In many cases, you want variable values to remain intact across several invocations of a program. Such persistence allows you to log into a database server only once, keeping a connection open through the duration of many HTTP requests.

This raises the question of what happens to variables you define in an Embperl document—do they also keep their values across invocations, or do they disappear? The answer is that each Embperl document is processed in its own package, and the variables defined in that package are reset by default upon each invocation. However, variables defined in other packages are kept across invocations. The following Embperl document demonstrates how this works:

```
<HTML>
<Head><Title>Current time</Title></Head>
[- $counter++; -]
[- $remain::counter++; -]
<Body>
<P>This is Embperl</P>
<P>Counter: [+ $counter; +]</P>
<P>remain::counter: [+ $remain::counter; +]</P>
</Body>
</HTML>
```

If you try this on your system, you may well discover that **$counter** always remains at 1, while **$remain::counter** is incremented with each invocation. However, if you are running more than a single copy of Apache, **$remain::counter** probably jumps around, as if several different copies of it were being incremented. This is indeed the case, since each copy of Apache is running its own copy of mod_perl and Embperl. If you rely on persistent variables across invocations, remember that a given user might connect to more than one copy of Apache, and you cannot rely on the same copy always being available to the same user.

However, persistent variables can be useful when making connections with other than the user's computer. In particular, DBI (the Perl database interface) can take advantage of this with the **Apache::DBI** module. This module opens a connection to a database server when it is first invoked, and then continues to use that connection throughout the life of the Apache process, immediately sending each query to the database server. Because the persistence is between Apache and the database server, it works regardless of whether a user connects to the same **httpd** process each time.

When defining subroutines inside of Embperl documents, it's probably best to use another kind of square brackets, with exclamation points as the special characters. Square-bang brackets ([! !]) are the same as square-minus brackets, except that the Perl code contained within is executed only upon the document's first invocation. If you are running Embperl under mod_perl, defining subroutines inside of square-bang brackets means they will be defined and compiled a single time, further increasing the speed of your program.

Embperl Meta-commands

Finally, we come to square-dollar brackets ([$ $]), which allow you to enter Embperl meta-commands. These meta-commands, as you might imagine from the name, are actually part of a small programming language with which you can tell Embperl what to do.

Meta-commands allow you to make sections of HTML and Perl conditional, or to loop over them a given number of times. The same tasks could be performed inside of a normal Embperl block, since Perl is a full-fledged programming language and can handle conditionals and looping just fine. But by using the Embperl meta-commands, you can place even more HTML outside of the Perl blocks, making the Perl blocks somewhat smaller and easier to read.

For example, let's say we run a web site that requires registration. Assuming we have a function called **&is_registered** that returns “true” or “false”, depending on whether a user is registered with our system, we could print an appropriate greeting with the following code:

```
[+ &registered($user_id) ? "You are known" : "You are
unknown" +]
```

Once you start to deal with the formatting associated with those strings, the menus you might want to display for new users and the personalized home pages that registered users should see, the block of Perl inside of square-plus brackets becomes quite large. It's thus easier to use square-dollar brackets and Embperl meta-commands:

```
[$ if &registered($user_id) $]
You are known, your registered home page:
        [+ &output_home_page($user_id) +]
 [$ else $]
 Welcome, new user! We would like to ask you a few
 questions:
 [+ &output_questionnaire +]
[$ endif $]
```

The above, which I have indented in the style of a programming language, is easier to understand than a large block of Perl code. It is also more easily understood and modified by non-programmers on your site, who can clearly see the difference between HTML and other items.

Automatic Table Generation

Embperl has many features, far too many to describe here. My favorite feature is its ability to create HTML tables automatically, filling them in as necessary. Embperl looks for the beginning of an HTML table, marked with a **<TABLE>** tag, before filling it in. In order to do this, Embperl expects you to use a number of “magic” variables within the table. You can set the exact behavior with Embperl's **$tabmode**, but the basic idea is that within a table, **$row** (a magic variable) begins at 0 and increments until it reaches **$maxrows** (another magic variable). When an expression within the table returns “undefined”, Embperl exits from the table loop and stops incrementing **$row**. We can thus get a nicely formatted listing of environment variables with this code:

```
<HTML>
<Head><Title>Environment</Title></Head>
<Body>
[- @keys = sort keys %ENV -]
<Table border=2>
<tr>
[- $index = $row -]
<td>[+ $keys[$row] +] </td>
<td>[+ $ENV{$keys[$index]} +] </td>
</tr>
</Table>
</Body>
</HTML>
```

Notice how we first defined each array outside of the table definition. We then used **$row** (which is incremented automatically by Embperl) to retrieve each element from **@keys**.

Using the magic table fill-in procedure can be extremely powerful, but it requires you to change your programming style somewhat. Nevertheless, the potential uses for it in database applications are tremendous, since it greatly cuts down on the amount of necessary coding.

If you look at the list of environment variables, you might notice **QUERY_STRING** is unset. When invoking programs, **QUERY_STRING** is normally set by appending a question mark (?) and a string to a URL, but there is no reason why we cannot use the same syntax with Embperl documents, as in **http://localhost/embperl/env.html?foo**.

If the above environment-printing Embperl file is called env.html, then invoking it with the **foo** parameter should give **QUERY_STRING** a value.

Indeed, we can even use Embperl documents as the “action” of a CGI program. Grabbing values from the %fdat hash, Perl blocks within our document can retrieve form values, use them, and even construct a document based on them.

Embperl does require a slightly different style of programming than is usual in Perl. Typically, Perl is written in blocks of code, with each code returning a value. Embperl is much terser, with pairs of square brackets occurring much more often than Perl's curly braces. Of course, the style presented in the Embperl documentation and the above examples does not have to be your own; you can put entire Perl programs between square brackets.

The trend seems to be toward using templates and databases to design web sites, with more and more products appearing on the market that claim to do such things. The combination of Linux, Apache, mod_perl and Embperl not only makes for a cost-effective solution, but also a powerful combination of programming tools that is hard to beat. Next month, we will look at Embperl a bit more closely, and learn how we can use it with databases to easily create personalized home pages.

Resources

Reuven M. Lerner is an Internet and Web consultant living in Haifa, Israel, who has been using the Web since early 1993. In his spare time, he cooks, reads and volunteers with educational projects in his community. You can reach him at reuven@netvision.net.il.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #54, October 1998

Readers sound off.

KHello

It has been pointed out to me that KHello no longer compiles correctly. This is because the KDE API has changed since the time my article “A First Look at KDE Programming” (August, 1998) and program were written. An updated version of the khello source code can now be found at http://www.chaos.umd.edu/~dsweet/KDE/KHello and in the tar file ftp://ftp.linuxjournal.com/pub/lj/listings/issue52/2653.tgz on the *LJ* FTP site.

—David Sweet dsweet@glue.umd.edu

UNIX Servers in a Notebook

I have thought about running Linux on an Intel notebook. As I began to look for notebooks which support Linux, I found a site that sells UNIX Server notebooks at http://www.tadpole.com/. They have both SPARC and Alpha notebooks. I was quite impressed. A SPARC20 model of the notebook supports Linux as well as an Alpha version.

Anyway, I thought your readers might like to know about this site. Maybe one day we will see a review of these notebooks in *LJ*.

—Robert Binzr binz@swconnect.net

If you get one, maybe you could do the review. —Editor

E-mail Address Correction

I just noticed that the article I wrote entitled “Linux Hits The Big Leagues” was printed with an incorrect e-mail address for me. The address should be altered to samw@wwa.com from samw@www.com. Thanks.

—Sam Williams samw@wwa.com

Red Hat 5.0

Simon Maurice's letter in the June *Linux Journal* leaves me speechless, so I have to type this response.

He accuses Red Hat of delivering a “truly bad release” and, as evidence, draws our attention to the errata list that includes listings of bugs for which no fix is yet available. Surely openness about what's fixed and what's not is at the heart of the Open Source movement. To go on and claim that Red Hat is behaving in a very Microsoft-like way defies both logic and the day-to-day experiences of system administrators around the world.

One could claim that Red Hat does not write the software containing these bugs, they merely package it, but in truth they do far more than that. Red Hat does a good job in maintaining their distribution and keeping it current. Their pioneering work with glibc (alongside the Debian project) is just one example of a benefit to the wider Linux community. If Mr. Maurice doesn't want to download a whole RPM file, he is welcome to download the source of any package and track patches from its maintainer. Nothing in the Red Hat distribution forces him to use RPMs—most users find them extremely convenient.

—Grant McLean sisl@ihug.co.nz

July 1998 Issue

I just finished reading the July issue, and I must say that it will most likely go on the shelf, never to be read by me again. I found no useful article in the entire issue. Don't get me wrong, I am all for Linux success stories, but most of the articles were so technically dry that I lost interest after a paragraph. I have been a subscriber since January 1998, and this is the first issue to achieve the title of “useless”. That being said, I love *LJ* and wish to see the ongoing improvement of both Linux and *LJ*. Thank you for hearing my whine.

—Griffin Caprio griffinc@ameritech.net

NIS Comments

I have a couple of comments on an article in the June 1998 issue: “Introducing the Network Information Service for Linux” by Preston Brown.

At the time of writing (February 1998), the latest version of Red Hat was 5.0, not 4.2, so the remark about an older version of ypserv in 4.2 has no value. What I do find a little odd is that 5.0 (released December '97) shipped the same version of ypserv, considering 1.2.5 had been out since mid-October. Currently, though, Red Hat 5.1 ships 1.3.1, which was (presumably, considering we now have 1.3.2) the latest one when they closed the distribution (June 1998).

The /contrib/hurricane directory contains software for Red Hat 5.0 (code name Hurricane, a glibc-based distribution) that can't be used (except in a few trivial cases) on libc5-based systems, such as Red Hat 4.2.

I'd like to know which features I missed by not having ypbind, but the article doesn't give this information; I can only say that all my programs worked flawlessly.

domainname was not invoked in Red Hat 4.2, but is present in Red Hat 5.0 (see the comment above on release dates).

—Andrea Borgia bab0069@iperbole.bologna.it

Comments on July

As a new convert to Linux, I am very satisfied with my subscription to *LJ*. I have comments about two of the articles in the July issue, one positive, the other less so.

Thank you for the avalanche predictor model by Richard Sevenich and Rick Price. It reminded me—at an optimal time—of the power and utility of fuzzy logic to many of the real-world problems we deal with on each project. I've applied fuzzy logic to a GIS (Geographic Information System) for land use decision making support. But, in the press of too many things to do and not enough time, I had forgotten about this powerful set of tools. Thanks to this article, I have a solution to at least one environmental problem which affects the mining industry. Since we use Linux, the model will be developed and run on this platform. If there's demand, we'll make it available on Windows 95, too, using Tcl/Tk for the user interface.

On a less positive note, I need to ask why Richard Parry's article on position reporting using GPS and ham radio was published. After reading the article, I still have the unanswered question, “So what?”

We use GPS data extensively in our field work. We use it to delineate wetlands, orthorectify aerial photos, measure mines and quarries for modeling of storm water runoff and reclamation planning, locate wildlife nests/burrows and other phenomena of the world out of doors. We also know that many surface mines use GPS technology to dispatch haul trucks and maximize operations, many police and fire vehicles are equipped with GPS/GIS systems for emergency response, and at least one long-haul trucking company (Schneider, with the orange tractors and trailers) uses the technology to increase their efficiency, customer service and driver satisfaction. Given all this, what is the value of having one's vehicle location transmitted to a network of amateur radio stations? I was a licensed ham operator (many, many moons ago), so I'm certainly not belittling them.

What I'm saying is that this article appears to be more of a reporting on, “we know how to do this, so we will,” than a solution to a real problem. If I've missed something in the article, please do let me know. Each of the other five articles related to the issue theme report how Linux facilitates solving a problem. This one article doesn't appear—to me—to fit that mold.

Regardless, kudos are due to all of you for a very useful information source (including the ads). I'll be buying a SCSI adapter from one of your advertisers tomorrow, since my relatively new (but shortly out of warranty) HP/CMS tape drive has decided to die.

—Dr. Richard B. Shepard rshepard@appl-ecosys.com

I thought Mr. Parry's article was fun; not every article needs to address a “real” problem. If I had a GPS article describing some of the ways it's used that you mention, I'd have been happy to publish it too —Editor

July BTS

In the “Best of Technical Support” column in the July issue, Mark Bishop responded to a question titled “Editing motd and issue”. Mr. Bishop forgot to point out that the Slackware distribution, by default on startup, overwrites the /etc/motd and /etc/issue files. To change this, one must comment out (#) the commands that overwrite the /etc/motd and /etc/issue files. Check out the /etc/rd.d/rc.local (I think) file in Slackware to make the changes. After doing so, you can edit the /etc/motd and /etc/issue files without fear of them being overwritten.

—Andrew Dvorak andrew_dvorak@IName.com

You have contacted me about the Slackware distribution overwriting on bootup. Not all distributions do this, however, and I believe that hasn't always been the default behavior of Slackware. I wish I had the time to keep up with all the distributions, and I plan on installing Slackware on my new machine now that a new version has been released (too bad it's not based on libc6). Thank you for pointing out that not all distributions are created equal.

—Mark Bishop mark@bish.net

Removing Files and Security

I am writing in response to Dave Lutz's (dlutz@smith.edu) letter on removing files and security. In it he discusses security issues presented by IRC and lynx, along with “hacker flags”.

He discusses how IRC is a security risk. IRC has been around for many years, and is the main form of chat used by people on the Internet. The act of being on IRC does not become a security risk. He cites the trading of “warez” and pirated software as an example. Web pages and other forms of chat are used to spread those, and they are not security risks. He also mentions the “eggbot”. The proper name of this is “Eggdrop”, and as one of the developers of this program, I can assure you that, if properly configured, the bot poses no security threat.

His letter also mentions how lynx can be used to bring in malicious files. Any program that can transfer files, be it ftp, mail, lynx or IRC, can be used to bring in files. Disabling them is not the proper way to secure a system, as this only hides the problem—it does not solve it. Several key issues should be looked into when securing a system.

One of the main issues would be setuid root programs, that is, programs which run under root permissions. Several distributions, especially older ones, come with setuid root programs which can be used to gain root access on the system. These include SuperProbe, xterm and others. The command **find / --perm +4000** can be used to list setuid root programs. If a program doesn't have a specific need to be root (try **chmod a-s** ***`program`*** and run it as a normal user), you should probably remove the setuid bit to be safe.

Several security lists are available which detail security problems as they are found, one of the main ones being Bugtraq. To subscribe, send mail to listserv@netspace.org with “subscribe bugtraq” as the subject.

Once again, security through obscurity can work for only so long. If the person wants into your system, the methods described by Dave will not prevent them from getting in.

—Jason Slagle jslagle@toledolink.com

Errors in July Article

There is an error in your July article entitled “Encrypted File Systems” by Bear Giles. On page 67, in the first column the following commands appear:

```
   mount /dev/fd0 /mnt -text2,loop,encryption=idea
   mount /dev/fd0 /mnt -text2,loop,encryption=des
```

They should read:

```
   mount /dev/fd0 /mnt -text2 -o loop,encryption=idea
   mount /dev/fd0 /mnt -text2 -o loop,encryption=des
```

That's it. You people do a great job. Keep it up.

—Steven J. Hill sjhill@plutonium.net

Linux and Informix

Phil Hughes

Issue #54, October 1998

The availability of Informix SE for Linux was announced at the International Informix Users Group conference, July 22-24, 1998.

I spent the last three days (July 22-24) at the International Informix Users Group meeting. What I saw and heard were some of the most significant announcements related to Linux yet. Why? For a host of reasons.

First, Informix didn't announce that something would happen in the future—they announced and delivered on the same day. Informix SE is available for Linux now. In fact, at the conference, you could pick up a free copy of Informix SE for Caldera or S.u.S.E. Linux at the Linux pavilion. (If you weren't there, check out http://www.informix.com/informix/products/lx.html for download instructions.) Also at the Linux pavilion were Caldera, Linux International, *Linux Journal*, Red Hat and S.u.S.E. Caldera, Red Hat and S.u.S.E. were giving out copies of their OS, and we were giving out *Linux Journal* issues. That meant you could pick up everything you needed at the show to create a Linux/Informix SE system—all for free.

A word of explanation is in order about which versions of Linux will run Informix. Reality is that it would probably work fine on Slackware and Debian, as well as Caldera and S.u.S.E. Red Hat has recently upgraded to glibc and is buggy enough that Informix would not run properly. Within the next few months, glibc should be working properly and it will be time for everyone to move to it—distribution vendors as well as applications vendors.

The second important thing about the announcement is that it immediately brings applications to Linux. Apropos Software makes a retail point-of-sale system. As of the show, they started shipping their 1.2 million-line Informix application on Linux. By the time you read this article in *Linux Journal*, I expect there will be many more vendors who have announced applications running on Linux/Informix.

The third important thing is how serious the Informix management is about this port. At the press conference, the Linux port was the first announcement. They then fielded a series of questions from the press. Their responses showed me that Informix knew a lot more about Linux than the press did. For example, one journalist from France asserted that this Linux announcement must be a “marketing trick”. Diane Fraiman, VP of Marketing for Informix, assured him that it was not and went on to explain how a Linux port benefited the average Informix user and VAR.

More specifically, here are some answers from questions asked at the press conference:

- They did the port because of pressure from user group members.
- The Informix SE port would give new life to VARs with a current product.
- Informix now has a trained Linux staff, so they could port another product such as the Informix Dynamic Server in 90 to 120 days.
- When questioned about the maturity of Linux in the enterprise market, Mike Saranga, Senior Vice President of Product Management and Development, said, “We have been waiting for two years for NT to mature. NT is starting to get there.” He went on to say that Linux is maturing rapidly.

I have left the best for last. This is a universal message about why Linux is the next platform to which many vendors will port.

I met with Stephen Lambright, Director of Server Product Marketing in a one-on-one session. I set up the appointment before I went to the press briefing, because I didn't expect much coverage of the Linux port. Clearly, I was wrong, but I did have one big question left.

I asked Steve how hard it was to port Informix to Linux. He told me that they just typed `make`. There was not a single line of code changed to make Informix SE run on Linux. He went on to say they did spend a lot of time testing the port, and everything still worked with no changes. I asked around, and apparently the port began with the Solaris version of Informix SE.

This one item is the most important happening for Linux—it helps give us an idea of how much work it will be to do the port. The time has arrived for all of us to start asking vendors to port their software to Linux.

If you haven't yet, check out the Linux Software Wish List on the Linux Resources web page (http://www.linuxresources.com/). The more you put on that page, the more ammunition we have when we contact vendors to encourage Linux ports.

Look for a review of Informix SE in *Linux Journal* next month.

Archive Index Issue Table of Contents

Advanced search

Virginia Power Update

Vance Petree

Issue #54, October 1998

Mr. Petree brings us up to date on events at Virginia Power, telling us about its Linux substation controllers and new data monitoring system.

If you're a longtime *Linux Journal* reader, you've probably read many exciting tales of Linux's success in the real world. Perhaps, after a relaxing evening hacking a device driver, or building a new kernel, or trying to memorize the Emacs commands to change fonts in a LaTeX buffer, you've sat on the edge of the bed, hand on the light switch and mused to yourself, “I wonder what those Virginia Power guys are up to these days?”

“What's that, dear?” your Significant Other murmurs sleepily (or, depending on your lifestyle, woofs or meows from the foot of the bed).

“You know, those intrepid fellows who used Linux to build a distributed data collection and archiving system (*Linux Journal* #9, January 1995), and a dial-up SCADA (supervisory control and data acquisition) system to interface to their existing SCADA system (*Linux Journal* #10, February 1995).”

“Oh—*those* Virginia Power guys ...”

“Yes. I wonder if they're still using Linux, if it still meets their needs, and if they're doing anything new and exciting ... ”

Well, as one of those Virginia Power guys, I can say quite happily: are we *ever*! I guess we've been so busy coding these past couple of years that we haven't had time to document what we've been up to, but I'll try to make up for that lack (at least a little) in this article. I'll describe two new applications in which we're using Linux—one already complete, installed and working wonderfully, and the other which, when completed, will be one of the largest computer systems of any type at Virginia Power, and possibly one of the largest systems using Linux anywhere.

The first application I'll discuss is a natural outgrowth of our earlier Linux applications—especially our dial-up SCADA system. I won't recount all the details of that system (see *LJ* #10 if you're interested)—the important thing to remember is that the dial-up system retrieves status and analog data *from* and sends control commands *to* remote devices installed in substations and on pole tops.

In larger substations, a device called a remote terminal unit (RTU) is usually employed as the primary remote device. The RTU serves as a data concentrator, collecting status and analog data from various other devices in the station and providing a central location from which the data can be retrieved remotely and to which control requests can be delivered.

RTUs are somewhat limited in processing power and expandability, and usually only interface with a limited number of other monitoring and control devices. As a result, one of the current trends in substation design is employing a computer as the station data concentrator. This approach allows not only greater flexibility as far as the types and numbers of devices which can be connected, but also provides a general-purpose software environment wherein some monitoring and control algorithms can be executed locally within the substation.

As you might expect, substation controller systems are commercially available, many implemented using MS Windows. However, these systems are expensive, difficult to administer remotely and contain a plethora of software gingerbread and geegaws (also known as click-and-drag disease) having nothing to do with monitoring devices in a substation.

Figure 1. Dial-up SCADA System

In the Operations Engineering group, we realized that our dial-up SCADA system (see Figure 1) provided the same functionality as those commercial substation controllers, *if* you eliminated the dial-up system and moved the PC and the translator device out into the substation (see Figure 2). The translator device is necessary to communicate with our existing SCADA Master control computers using an ancient bit-oriented protocol (over 1200bps leased lines, no less). However, these control computers are due to be replaced with a new computer system over the next couple of years—more on that later.

To cut to the chase, moving PCs to the substations is *exactly* what we did. At the time of writing this article, half a dozen Linux-based substation controllers are installed and working around the clock, with more in the planning stages.

Figure 2. Substation Controller System Setup

The basic design of the substation controller is pretty straightforward. For each type of special equipment which performs the actual monitoring and control in the substation, a specific *protocol task* is written (for compulsive coders, this is the fun part) which handles all the details of data retrieval and control execution. The devices which perform the actual monitoring and control are usually referred to as IEDs (Intelligent End Devices), since they have a certain amount of intelligence built into them and sit at the end of the data retrieval and control path. Some IEDs communicate via serial lines; others use specialized local area network protocols such as **ModBus+**.

Another protocol task communicates with the translator device, which in turn communicates with the SCADA master computer. A database management daemon coordinates the activities of all these protocol tasks and also maintains shared memory partitions which contain the actual data. With all of these components taken into account, a typical substation controller setup looks like the one in Figure 3.

Figure 3. Substation Controller Design

In most cases, the substation controller computer is rack-mounted along with the rest of the equipment in the substation “control house” (a small building intended mainly for protection from the elements). It has no keyboard and no monitor (try *that* with your Windows box). All system administration is performed remotely via dial-up login; database and program updates are distributed using UUCP. In some cases, the installation includes a touch-screen monitor to provide a local operator interface, consisting of an annunciator panel (see Figures 4 and 5) and even an interface for performing device controls (see Figure 6).

Figure 4. Substation Annunciator Panel

Figure 5. Substation Panel

Figure 6. Substation Controls

By the way, an *annunciator panel* is just a hierarchical display of alarm points, in which an alarm at a higher level means one or more points at a lower level are in an alarm state. The annunciator panel initially displays the topmost level of alarms as a grid of labeled boxes (green if all lower alarms are normal, red if at least one is in alarm). A substation technician can touch one of the boxes to display the next lower level of alarms; each one of *those* alarms can consist of additional individual alarms, etc.

These substation controller systems are highly flexible. If more serial ports are needed to talk to additional devices, we need only augment the serial multiport card or add another multiport card. If new types of devices are to be connected, we need only write a new protocol task and possibly a new device driver for an interface card.

How reliable are these systems? All of our substation controller systems have functioned extremely well from the moment they've been powered up in the field. We've had a few application software bugs and glitches, but the system software has never failed nor caused us a single problem through thousands of hours of continuous uptime.

I stopped worrying long ago about the robustness of Linux—to be honest, Linux and its associated tools and support software from the Free Software Foundation and elsewhere comprise the most reliable system I have ever used on any hardware platform. I've heard horror tales from users of other operating systems: blue death screens, exhausted resource limits, quirky compiler bugs and so on. I shake my head in rueful irony, then swivel my chair back to my Linux box to get some work done. I can't imagine putting any *other* system out in the middle of the wild and woolly real world and expecting it to run for weeks and months and years without fail.

As far as cost-effectiveness goes, new controller systems generally cost only as much as the hardware, with sometimes a little software overhead if a new protocol or device driver needs to be developed. No commercial system can hope to come even close to that low a cost—thanks to the freely distributable nature of Linux. Since our Operations Engineering group is responsible for finding cost-effective solutions for power system monitoring and distribution needs, Linux is just this side of a miracle. In a very real (albeit small) sense, Linux helps us keep down the cost of distributing electricity. With every new substation controller installed, customers who've never even heard of Linux can benefit from all the hard work and loving craftsmanship the Linux developers and maintainers have invested in their system.

These two important points—reliability and low cost—lead naturally into a discussion of our *other* big Linux project: a replacement system for the current network of SCADA Master computers. The SCADA Master computers are the systems which scan all of the RTUs and IEDs mentioned above (several hundred at last count). These computers monitor power system conditions, generate alarms when abnormal conditions are detected, and provide system operators with summaries of power grid information, one-line diagrams of substation layouts and control interfaces for remotely operating breakers, capacitors and other field devices. The systems also run closed-loop feedback control programs, which automatically respond (usually via device controls) to changing system conditions.

Currently, the SCADA Master computers are a network of six PDP-11/84 computers which have just about reached the end of their usefulness—they've reached their limits for CPU power, installable RAM and so on. The user interface is a creaky mixture of specialized keyboards with banks of function keys and a character-based graphics terminal with unchangeable little symbols and line segments for drawing substation one-line diagrams. All of these features were quite new and progressive in the early 1980s, but are far from flexible enough for the present or foreseeable future.

As with our substation controllers, we went the commercial route first when we started looking for a new SCADA system. We reviewed the offerings of about a dozen vendors. Alas, since a SCADA system is similar to a factory automation system or even an aircraft simulator, many systems we reviewed were derived from these types of systems. As a result, they contained many features and add-ons which made no sense for the way our operators used SCADA; plus, they were expensive; plus, we usually couldn't get access to source code. (We've been spoiled by Linux.) At least one vendor offered to put the source code in *escrow*—about as useful as being told how delicious a chocolate cheesecake is without being offered a slice.

Not least of all was the issue of retraining our operations personnel (not to mention ourselves) on a completely new system. Monitoring the electric distribution grid is a 24-hour-a-day job, so we couldn't just shut down shop for a couple of months while we came up to speed on a new boatload of software. After all, our goal was to reliably monitor and control the power system, not necessarily to learn an entirely new way of opening a circuit breaker or logging an alarm. (Similar to click-and-drag disease is the curious notion that a new or different way of performing a task is *automatically* a better way.)

We thought long and hard about what we *really* needed: a cost-effective, flexible, scalable, reliable SCADA system replacement that wouldn't require extensive retraining for ourselves or our system operators, and that wouldn't include extra software gadgets for which we would have no use.

Meanwhile, our Linux systems continued to run quietly day in and day out: performing dial-up data retrieval, monitoring scores of devices in substations. The substation controllers, in particular, were almost embryonic SCADA systems, with data retrieval, database storage and archiving, and a user interface with one-line schematics and controls.

Of course, several important pieces were missing from the substation controllers which would have to be supplied to turn them into a full-fledged SCADA system—but after you've spent enough time in Emacs, you tend to think you can accomplish anything. So we examined issues of scalability, figured out exactly what extra pieces we needed and whether we could develop them on a realistic schedule, and put together a presentation for our management.

Surprisingly (or perhaps not so surprisingly, given our track record with the other cost-effective Linux systems), our upper-level management gave us the green light. Suddenly we had *plenty* of work to do, with an implementation target date of December, 1998.

I can't adequately describe how exciting (and terrifying, in some respects) this project is. Some of the features of our design:

- A private high-speed (100 Mbps) wide area network connecting all our main machines, separated from our corporate network by a firewall
- 500 MHz DEC Alpha database servers
- High-speed Intel front-end processors to handle RTU scanning and database retrieval
- Multi-headed operator workstations running X
- A distributed shared-memory database to transparently share information among all servers and workstations
- Three regional operating centers, a duplicate center at our Grayland Avenue office, scores of workstations, and many district centers (some connecting via Ethernet, others on demand via PPP/SLIP)
- Everything running Linux

A general overview of our system, which we've named swSCADA for SkunkWerx SCADA (smile) is shown in Figure 7.

Figure 7. swSCADA Overview

By the time our implementation is finished (as mentioned above, our installation begins in December, 1998 with a new system in the Eastern/ Southern regional center, with the remainder of the centers being phased in by the end of 1999), we will have a network of around four dozen Alpha and Intel boxes, running 24 hours a day, 7 days a week. This probably isn't the *largest* network using Linux systems exclusively, but it certainly puts the lie to those armchair critics who claim large corporations are unwilling to use Linux in mission-critical situations. Monitoring the electric power distribution grid is a mission-critical situation for a power company, and not only are we *willing* to use Linux, we embrace it wholeheartedly (and admittedly, somewhat evangelically). Its robust quality and freely distributable nature will save our customers money, provide them with top-drawer service, and give our shareholders more of a return on their investment. In today's bottom-line business environment, *those* sorts of arguments matter.

I was thinking the other day about Linus Torvalds' ultimate goal for Linux: “world domination”. As we enter the 21st century, every light, CD player, television, toaster, PC and hair dryer in central Virginia and part of North Carolina will be under the benevolent, watchful eye of vigilant Linux swSCADA systems. That's not world domination *yet*—but it's a start!

Vance Petree has been trying to write bug-free code since 1969, although he has yet to succeed. In his spare time—well, he doesn't really have any spare time right now, but when he does, he listens to loud symphonic music and reads too much science fiction. He lives with his beautiful wife (a professional artist) and delightful daughter (a professional toddler) in the antebellum backwash of Richmond, VA. He can be reached at vwp@vancpower.com.

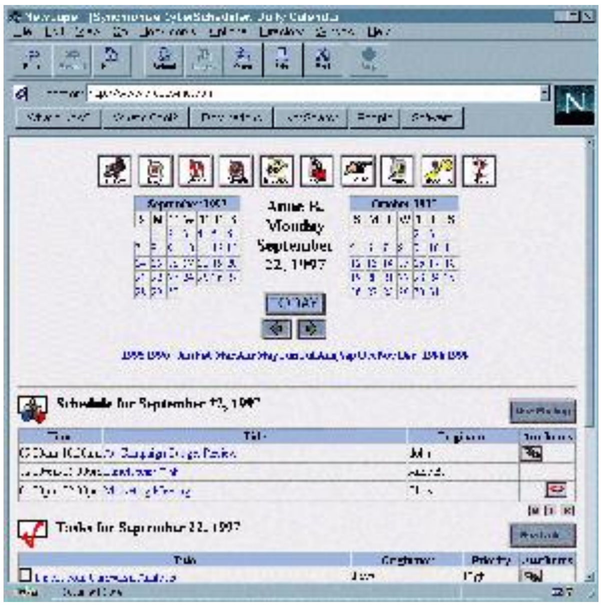
Advanced search

New Products

Amy Kukuk

Issue #54, October 1998

CyberScheduler for Linux v2.1, S.u.S.E. 5.3, Informix SE on Linux and more.



CyberScheduler for Linux v2.1

CrossWind Technologies, Inc. has announced the availability of CyberScheduler for Linux v2.1. CyberScheduler for Linux is an Internet-based calendar and scheduling solution for workgroups. CyberScheduler's advantage is its low cost and ease of installation. The product has been packaged to support Apache web servers running on all major Linux distributions such as Red Hat, S.u.S.E. and Slackware. The installation process has been significantly streamlined with the new version 2.1, supporting both RPM and TAR installations. CyberScheduler for Linux retails at $49.95 US per end user.

Contact: CrossWind Technologies, 140 DuBois Street, Suite D, Santa Cruz, CA 95060, Phone: 408-469-1780, Fax: 408-469-1750, E-mail: info@crosswind.com, URL: http://www.crosswind.com/.

S.u.S.E. 5.3

S.u.S.E., Inc. has announced the latest release of its flagship product, S.u.S.E. Linux version 5.3. S.u.S.E. 5.3 includes an option for quicker and easier installation, support for FAT32 file systems and the ability to run and install both glibc-based and libc-based packages. Linux software titles included in version 5.3 are Netscape Communicator 4.05, the KDE window manager and the Gnome desktop environment. Version 5.3 also includes S.u.S.E.'s own Xfree86 3.3.2.3 XServer. S.u.S.E. Linux 5.3 is scheduled for a tentative release date in early September at a price of $49.95 US. It may be purchased via subscription service at a price of $34.95 US.

Contact: S.u.S.E. LLC, 458 Santa Clara Avenue, Oakland, CA 94610, Phone: 510-835-7873, Fax: 510-835-7875, E-mail: info@suse.com, URL: http://www.suse.com/.

Informix SE on Linux

Informix Corporation has announced the release of Informix SE on Linux. Informix SE is an SQL-based database engine for small- to medium-range applications. It is a solution for businesses that want the power of SQL without the complex database administration requirements. Linux application developers are now able to download a free developer's kit that includes Informix SE, ESQL/C for Linux, Informix's SQL toolkit and I-Connect (the runtime version of ESQL/C). Informix SE and ESQL/C in Linux is available from Caldera and S.u.S.E. on the Intel platform.

Contact: Informix Software, Inc., 4100 Bohannon Drive, Menlo Park, CA 94025, Phone: 604-926-6300, URL: http://www.informix.com/.

GO-Global 1.5

GraphOn Corporation has announced the release of GO-Global 1.5. The product, a thin-client PC X server, has added functionality and platform support, including first-time support for Linux from Red Hat and Caldera on the PC architecture. The product is a native 32-bit, X11R6-compliant, distributed thin-client X server built around client/server architecture. This architecture delivers performance from low-powered laptops to high-powered Pentium machines. GO-Global 1.5 includes fast, dial-in, low-bandwidth throughput for PC to UNIX/X connections, simple installation and one-click connection.

Contact: GraphOn Corporation, 150 Harrison Avenue, Campbell, CA 95008, Phone: 408-370-4080, Fax: 408-370-5047, E-mail: sales@graphon.com, URL: http://www.graphon.com/.

SiteEater 1.0

SFS Software has announced SiteEater 1.0, a pure Java software utility for retrieving, presenting and archiving Internet sites. SiteEater makes it possible to download entire web sites from the World Wide Web or specific types of files such as programs, images or videos. The pioneering utility works on FTP and HTTP and can work on any Java-supported operating system. SiteEater enhances SFS Software's popular DocFather Professional 2.0 software and gives users the ability to create a full-text search index for any site. The product is priced at $49 US for a single user license and is available directly from SFS Software's web site.

Contact: SFS Software, Martin-Luther-Ring 31, 98574 Schmalkalden, Germany, Phone: +49-172-471-4485, E-mail: info@sfs-software.com, URL: http://www.sfs-software.com/.

IntraStore Server 98

Control Data Systems, Inc. has announced a Linux version of its IntraStore Server 98. The server and up to 250 user licenses can be downloaded from http://www.intrastore.cdc.com/. IntraStore includes messaging, web and directory servers. It offers superior messaging and application hosting capabilities, and its interoperability with existing systems enables workgroups to link LDAP directories, disparate mail and news clients, and otherwise stand-alone applications for immediate utility. Corporate users can share hosted applications and access IntraStore's functions (using IMAP4, POP3, NNTP, HTTP, SMTP, FTP and LDAP protocols) from most major mail clients and web browsers.

Contact: Corporate Communications Control Data Systems, Inc., 4201 Lexington Avenue North, Arden Hills, MN 55126, Phone: 612-415-2999, Fax: 612-415-4876, URL: http://www.cdc.com/.

LinuxCAD v1.55

Software Forge Inc. announced the release of LinuxCAD v1.55. LinuxCAD v1.55 includes hardcopy capabilities such as output to the LaserJet family of printers, output to DeskJet printers, PostScript black-and-white and color, HP-GL compatible plotters and LinuxCAD MS Windows print server. LinuxCAD v1.55 is $75 US.

Contact: Software Forge, Inc., Phone: 847-891-5971, E-mail: sales@softwareforge.com, URL: http://www.linuxcad.com/.

Empress RDBMS V8.10

Empress has announced the availability of Empress RDBMS V8.10, a comprehensive database application development solution. Spanning business to engineering applications, Empress provides one management system capable of handling multiple, diverse data types. In addition, options which allow the database developer to expand the functionalities of Empress are now available. These include object technologies such as persistent stored modules, a C++ host language interface and an Empress JDBC interface. The price of Empress RDBMS V8.10 varies depending on the platform, number of processors and users.

Contact: Empress Software, 6401 Golden Triangle Drive, Greenbelt, MD 20770, Phone: 301-220-1919, E-mail: info@empress.com, URL: http://www.empress.com/.

PCI-Wide/Ultra2 SCSI RAID Disk Array Controllers

ICP vortex announced the release of Wide/Ultra2 SCSI RAID Disk Array controllers for PCI systems, supporting RAID levels 0, 1, 4, 5 and 10. The new ICP controllers of the GDT-RD series are six models with 1, 2, 3 or 5 Wide/Ultra2 SCSI channels. The ICP controllers are based on a high-performance hardware and software architecture that contains Intel's RISC CPU. The Wide/Ultra2 SCSI channels can be operated either in Single-Ended (SE) mode or in Low Voltage Differential Signaling (LVDS) mode. LVDS operation with LVDS SCSI devices allows data transfer rates of up to 80MB/sec and a cable length of up to 12 meters. The controllers are PCI 2.1 compatible and enable transfer rates on the PCI bus of up to 132MB/sec.

Contact: ICP vortex Corporation, 4857 West Van Buren Street, Phoenix, AZ 85043, Phone: 602-353-0303, Fax: 602-353-0051, E-mail: info@icp-vortex.com, URL: http://www.icp-vortex.com/.

Cyclades-PR3000 with T1 and ISDN Interface Cards

Cyclades Corporation announced the availability of the T1/E1 and ISDN Interface cards for the PR3000 modular router. Delivering more than 50 MIPS of raw performance, the PR3000 can offer performance under the most demanding conditions. The T1 Interface Card has a built-in, full-featured DSU/CSU, allowing direct connection of the router to the T1 or fractional T1 line. An E1 model (for countries in Europe and elsewhere) is also available. The T1 and E1 Interface Cards cost $778 US each, and the ISDN Interface Card is $490 US.

Contact: Cyclades Corporation, 41934 Christy Street, Fremont, CA 94538, Phone: 510-770-9727, Fax: 510-770-0355, E-mail: sales@cyclades.com, URL: http://www.cyclades.com/.

Automating Tasks with EXPECT

Vinnie Saladino

Issue #54, October 1998

Mr. Saladino gives a quick introduction to EXPECT, a program to help you accomplish your remote tasks.

As the System Administrator for 18 VAXs and 6 HP/UNIX machines, I am always looking for an easier way to do things. One thing I always do is write shell scripts for UNIX and command procedures for VMS to automate some of my tasks. If I could run them from one computer, I could administer the world from one place without logging on to each computer to run each task I have to perform.

I first tried using VAX/VMS command files in order to TELNET to the other machines. That method was quickly dropped, as the remote computer's I/O response arrived before my command file could do an input statement. Logons hung up quickly. I then tried to TELNET from a UNIX script—the same thing occurred. This type of frustration is why an administrator's job is tough.

Hanging out in a Barnes & Noble bookstore and leaning on the UNIX shelves is always a good pastime—you look good, and you help dust off the books. Leaning is how I discovered *Exploring EXPECT*, an interesting title for a book with a back cover stating “automate TELNET, FTP, passwd...”. I bought the book —yes, I judge a book by its cover (sometimes).

The excellent book, *Exploring EXPECT* by Don Libes (published by O'Reilly & Associates) contains everything you need to know about controlling any spawned process. You can spawn a game and interact with it as easily as a remote TELNET session. The book is fairly long and very detailed, but don't be intimidated. There is an easy way to learn to write EXPECT scripts; I'll get to it shortly.

To make a long story short, EXPECT is a toolkit for automating interactive programs, such as TELNET and FTP, written by Don Libes. EXPECT has to be the greatest asset to system administrators in all of history. Finally, I can easily write and execute TELNET logon scripts and do whatever I wish using a script file from one machine.

EXPECT reads commands from a script file, spawns a process like TELNET, sends text from the script file to the TELNET process, saves every character returned from the TELNET session, and “looks” for known character strings that the user “expected”. The script can test for different strings and execute different code based on the results. TELNET scripts can have a lot of intelligence built into them.

The first day at work after buying the book, I logged on to one of my HP/UNIX machines, did a **man expect** and received the message “no entry found”. After a few more tries, I made a call to HP. EXPECT and Tcl, the language EXPECT is based on, are not installed and are not even supplied on the CD-ROM distribution media from HP. HP recommended I download Tcl and EXPECT from a web site and install them. I do not like to do this sort of operation on a production server. After delivering a few favorite curse words, I went to my desktop Linux system.

When I installed Linux, I carved out a 1GB partition, did a full Linux install and dual booted it with MS Windows. A two-minute reboot, and Linux was up and running. Again, I ran **man expect**. This time, I got a man page—a very long and informative man page.

“Introduction EXPECT” is a program that “talks” to other interactive programs according to a script. Does this mean I have it? I ran the command

```
find / -name expect
```

and got this result:

```
/var/lib/LST/installed/expect
/var/lib/LST/contents/expect
/usr/bin/expect
```

Oh yeah, I've got it—I love Linux!

Poking around /usr/bin with the command **ll | more**, I found **autoexpect**. What's autoexpect? It's not in the book, so I typed **man autoexpect** and received the following output:

```
autoexpect - generate an EXPECT script from watching a session.
```

This is a cool program. I can run a TELNET or FTP session to a VAX from Linux, and create an EXPECT script from the session that I can run any time I wish.

Nothing beats actually doing it. I logged on to a VAX with FTP, **put** a file and quit. I copied the man page for EXPECT to a file, then I put it on a VAX using FTP.

```
man expect | col -b > man_expect.txt
```

Next, I used FTP to connect to a VAX named ZEUS and put the file man_expect.txt on that machine.

```
autoexpect -f test1.exp FTP ZEUS
autoexpect started, file is test1.exp
connected to ZEUS
220 Zeus FTP Server
Name (zeus:vinnie): v12321
Password:
230 User logged in.
Remote system type is VMS.
FTP> put man_expect.txt
 ...
FTP> quit
221 Goodbye.
Autoexpect done, file is test1.exp
```

The file test1.exp is the script created by autoexpect. I can rerun this script any time I wish simply by typing **test1.exp** at the prompt. Examining test1.exp shows the details of the two-way conversation between the Linux machine and the VAX. Every character is saved in either a **send** or **expect** command. Even the password is saved, so care must be taken with these scripts.

I edited my test1.exp script to eliminate the comments. I also took out all the non-essential expect commands, leaving only the bare essentials. The script was short and sweet. The whole concept of EXPECT is quite simple—send commands as normally typed, such as the VAX FTP prompt:

```
expect -exact "FTP> "
```

TELNET was easy too, but the script created by autoexpect would not run. Why? The basic problem of getting two computers to talk to each other is that they never say exactly the same thing twice; in particular, time and date character strings are always changing. Don points this out in the man pages for EXPECT, and again in his book.

When I used TELNET to log in to my VAX, the first thing it returned to me after the “Welcome” message was the time and date. After I did a “Directory Listing”, the file names and total number of files were returned. These responses vary with time. Removing them from the autoexpect scripts causes them to rerun perfectly every time. I wrote this article as an introduction to EXPECT, and to show how easy it is to use. *Exploring EXPECT* goes into detail on all the different aspects of EXPECT—from how it all works, to programming two-way conversations in EXPECT and how to log selected EXPECT output to a file. Many things you never thought could be automated can be run while you sit back and enjoy reading *Linux Journal*.

Vinnie Saladino has been immersed in UNIX for 4 years, and VAX for 19 years. He has a BS in Electrical Engineering and an MS in Computer Science. Currently, UNIX and Oracle take up most of his time. He enjoys building, nailing, sawing and wiring, as well as working with stained glass. He lives with his two kids, two dogs and two computers. He welcomes your comments at saladino@idt.net.

Archive Index Issue Table of Contents

Advanced search

Advanced search

Mastering Kernel Modules with Caldera

David B. Nelson

Issue #54, October 1998

Mr. Nelson gives us step-by-step instructions for loading kernel modules, so we can keep our kernel lean.

You shouldn't have to read this article. The concept of Linux kernel modules is fairly simple. Unfortunately, information needed to compile, install and use modules is scattered over several HOWTOs, READMEs, and man pages. Plus, the files which need to be modified are in several obscure directories.

I finally wrote this cookbook approach to get myself, and you, started. Once you are up and running with modules, you can dig into the details later. I tested this material on an X86 processor running Caldera Open Linux 1.1, which is close to Red Hat 4.2. Your mileage with other processors and distributions may vary.

Why use modules? They let you compile a small, fast kernel, then install and remove device drivers on demand. Without modules, the Linux kernel could become bloated and resemble a certain commercial OS.

First, I recommend you compile a base kernel including all essential capabilities for your system without modules. I know this sounds like we are going backwards, but you don't want to lose the ability to boot up because you messed up your modules. The README in /usr/src/linux is your guide, but basically you execute the command:

```
make mrproper; make xconfig
```

(or **menuconfig** or **config**) to include all needed capabilities, then run:

```
make dep; make clean; make zImage
```

Save your kernel configuration to a file named kernelconf.base, in case you need to recompile. The xconfig menu prompts you to save and load configuration files. If you use menuconfig or config, the current configuration is in the file /usr/src/linux/.config; copy that file to kernelconf.base. If you configured too big a kernel, final compilation will fail. If this happens, execute **make bzImage** instead of **zImage**. Your compiled kernel will be in the directory /usr/src/linux/arch/i386/boot.

You might have made a mistake in compiling your base kernel, so don't throw away your old one. If you are running LILO, rename your new kernel to zImage.base and copy it to the location of your current kernel, usually / or /boot. Add a section to /etc/lilo.conf that lets you select either your default or base kernel on bootup. My lilo.conf is shown in Listing 1, minus some comment lines.

The important addition to lilo.conf is the last section (**#base kernel**) which tells LILO about your new kernel. Also, be sure lilo.conf has **prompt** and **timeout** lines. Now execute **lilo** and then reboot. LILO will pause, giving the prompt **boot:**. If you hit TAB, you will be given the choices **linux** and **base**. Enter **base** and your new kernel will boot. You may get complaining messages about bad module dependencies, but if your base kernel is complete, that shouldn't bother you. If something goes wrong, reboot and enter **linux** (or just wait through the timeout interval) and your old kernel will boot. Make sure you have a working base kernel before proceeding. With this approach, you never burn your bridges (or kernel) behind you.

If you don't use LILO, make a boot floppy for your base kernel. To do this, insert a floppy and execute **make zdisk**, instead of **zImage**.

You are now ready to compile a kernel with modules tailored to your system. Execute the same commands as above, but when you execute xconfig or menuconfig, pick some features to compile as modules. I suggest you experiment first by picking nice-to-have, but not-necessary, modules to add to zImage.base. Good choices might be printer support or floppy support (unless you are booting from the floppy). Save your configuration as kernelconf.mod in case you need to go back. Also, write down which modules you are compiling. To know exactly which modules are compiled, I suggest you move or delete your old modules (if any). The Caldera release includes a lot of modules. They are in /lib/modules/2.0.29. I moved my old ones into subdirectories rather than deleting them, in case I needed to back up. If you are working with a different release of the kernel, instead of subdirectory 2.0.29 you will have a subdirectory corresponding to your release number.

After executing **make zImage**, run:

```
make modules; make modules_install
```

As before, go (using **cd**) to the directory containing zImage, rename it zImage.mod and move it to the directory where LILO will look for it. Put a new section at the bottom of lilo.conf to let you boot this kernel with the label **modules**. If you don't use LILO, make another zDisk.

Now, execute **depmod -a**. This creates /lib/modules/2.0.29/modules.dep, needed by module utilities. Next, execute the following:

```
modprobe -c | grep -v '^path' > /etc/conf.modules
```

This command sets up another file needed by the module utilities.

Now reboot, choosing label **modules** at the boot prompt. Next, move to the /etc/modules/2.0.29 directory. It should contain a file with a very long name like the following:

```
#1 Tue Feb 11 20:36:48 MET 1997.default
```

This file is read at boot time by /etc/rc.d/rc.modules. [Debian and Red Hat use /etc/modules—Ed.] It contains a list of the default modules loaded when the kernel boots. You need to change both the name and the contents. Fixing the name is the hard part. In directory /etc/modules/2.0.29, execute the commands:

```
FILE=`uname -v`.default
cp "#1 Tue"* "$FILE"
```

This magic creates a file with the name that rc.modules will look for on bootup. The name is based on the time the kernel was compiled. If you recompile the kernel, you must repeat the magic.

Edit this file to contain just the modules you want loaded at bootup. For example, it might contain the lines

```
floppy
lp
```

which would load the floppy and printer modules, assuming you compiled them as modules. To get your editor to accept this file, you may need to put quotes around the name.

To load a module manually, execute **insmod** with your module name as argument. To remove it, execute **rmmod** with your module name as argument. To tell which modules are currently loaded, execute **lsmod**.

The best toy is **kerneld**; it automatically loads and unloads modules as needed. Assume you have compiled the floppy driver as a module. Check whether it is loaded by executing **lsmod**. If it is, remove it by executing **rmmod floppy**. Then execute **kerneld**. Now execute **mount /mnt/floppy** (or whatever mounts your floppy). Magically, kerneld installs the floppy module when needed. It will also uninstall modules which haven't been used for a while, keeping your kernel lean and mean.

You now know enough to experiment with modules without crashing your kernel on bootup. Read the Module mini-HOWTO, the kerneld mini-HOWTO, and the man pages for the utilities to become a real expert. Happy moduling!

Resources

David B. Nelson (nelson@er.doe.gov) manages scientific research at the U.S. Department of Energy. Before that, he earned his living as a theoretical plasma physicist. He started programming on the IBM 650 using absolute machine language and later graduated to CDC, DEC and Cray machines for his research, but thinks Linux is the most fun. He and his wife, Kathy, live near Washington DC; they enjoy tennis, skiing, sailing, music, theater and good food.

Best of Technical Support

Various

Issue #54, October 1998

Our experts answer your technical questions.

Authentication Failure

When I use Netscape from a Windows 95 PC to access my Linux 5.0 mail server, I get a prompt (as I should) for my login and password. However, when I type my password (or login) wrong, authentication fails and the server won't let me read mail, but it will let me send mail. That mail is sent with my name and address on it, which makes it easy for people to use my account and send mail using my name. Since I am also the administrator of that mail server, should I shut down **imap** and **pop3**? I don't want my users to have this problem too.

I have checked out PAM configuration, and imap uses the same library (pampwdb.so) as login and other critical services to authenticate users. So, what can I do to solve this problem and provide imap and pop3 service? —Diego A. Puertas Fernández, Red Hat 5.0

There is nothing you can do about it. What actually happens is that Netscape uses SMTP to send mail, and SMTP doesn't require any kind of session or login. Netscape (or any other mail client, for that matter) connects to the **sendmail** running on your server, gives it your piece of e-mail, and your server relays it to the Internet. Anyone can send mail as anyone else (especially on a Windows machine, since it doesn't provide the **ident** service and doesn't return a result that can be trusted). If you look in e-mail headers, most of the information (outside of the **Received** lines) can easily be spoofed. —Marc Merlin, marc.merlin@magic.metawire.com

Keyboard Troubles

I run a small web-hosting service in San Juan, Puerto Rico. The keyboard of one of my servers has stopped responding to Linux boots and I can access it only through TELNET. The server there has no response from the keyboard. My guess is that it was hacked as security in my network is just starting to be updated. Can you please help?

I've been told that by closing the unused ports I can have a more secure environment—how is that done? —Frank Nazario, Red Hat 5.0

The keyboard not working sounds like a hardware problem. Will the keyboard let you go into the BIOS setup?

As far as security goes, first of all edit /etc/inetd.conf and comment out anything you aren't sure you need, then restart **inetd** (with **killall -HUP inetd**). If it turns out you need something you commented out, just go back in to /etc/inetd.conf and uncomment it, then restart inetd again.

Then you'll want to kill any daemons you don't need. Be a little more careful here—make sure you don't kill off things like **atd** and **crond**. On the average server, you can (and should) kill off things like **lpd**, **nfs**, **portmap** and **smb**. Run /usr/sbin/ntsysv (a very handy utility included with Red Hat 5.x) and uncheck things that shouldn't be started. Then reboot or do:

```
for file in /etc/rc.d/rc3.d/K* ;
do $file stop ;
done
```

to make sure the things that shouldn't be running aren't.

The other important thing you want to do is keep up with updates from Red Hat. There have been some pretty major security holes announced recently, so you'll want to get all the updates on a regular basis. —Steven Pritchard, steve@silug.org

Rebel CD-ROM

My ATAPI CD-ROM won't eject or unmount in X—I get an error message saying the drive is busy. I tried closing everything but one X term and made sure the current directory wasn't in the CD, but it still says it's busy. Quitting X solves this problem. —John Vestrum, Red Hat 5.0

In my experience, this problem has been caused by a CD player running in a menu/button bar (such as AfterStep's “wharf” button bar) or a player that did not exit correctly after being used. Make sure there are absolutely no audio CD players running anywhere—if necessary, disable CD players you may have in a “wharf” type button bar—and see if that helps. —Erik Ratcliffe, erik@caldera.com

Modem Questions

I am about to install Red Hat 5.0, but would like to know if I need a new modem first so I don't lose WWW access. My current modem is a “Windows only” US Robotics Winmodem—yeah, I know—but it was cheap.

Do I need a new modem, or is there a driver available that enables use of this Winmodem under Red Hat 5.0? —Bill Brower, Red Hat 5.0

Sorry, but unfortunately you need a “real” modem. By “real”, I mean one that simply presents a serial interface to the hardware or plugs directly into a serial port. The problem with the “Winmodem” is that it does all the signal processing using the CPU of the host computer instead of having its own CPU to do the grunt work. I don't want to debate the relative merits of that process here, but suffice it to say that I will never buy a Winmodem nor will anyone I know. —Donnie Barnes, redhat@redhat.com

Poor FTP Speeds

I am getting very poor FTP speeds, but speeds of remote X applications are fine, 1.5MB/sec. I am using a Linksys Pocket Ethernet adapter on LPT1 and the de620 driver. Typing the command **more /proc/dev/net** shows “no lost”, and **colls packets... dmesg** shows “etho: Page out of sync! Restoring...” During FTP testing, I got a few “Illegal Packet size -4! Illegal Packet size 24!” messages when I changed the window size for that route by adding:

```
-net xxx.xxx.xxx.0 netmask 255.255.255.0
dev eth0 window X X= default,
8000, 800, 400, ...
```

The adapter seems to be working well from the X application testing, but FTP is useless. It starts fine (first set of packets) and quickly slows to a crawl, never to pass 2MB. Any ideas? —James T. Billups III, Red Hat 5.0

All I can think of is that you are running with a bi-directional printer port. Try changing this port back to “normal” (sometimes referred to as SPP) in your CMOS setup, or on your I/O card if it's not a built-in motherboard-type printer port and see if that helps. The source code for the de620 driver includes some comments that indicate this is the mode your printer port should be in when using this adapter. —Erik Ratcliffe, erik@caldera.com

Ugly Back Slashes

My question is about /etc/issue. I edited this file to show the name of my computer and it contains some backslash characters. I had to write these as \\ (double backslash) because one backslash is used as an escape character. But, when I access my computer through TELNET, both backslashes appear and look quite unpleasant. Is there any way of avoiding this? —Mihai Bisca

Your **getty** and **telnetd** parse the /etc/issue file in a different way. Create an /etc/issue.net file to provide a message for telnetd, while /etc/issue is reserved for local use. Naturally, this depends on the internals of getty and telnetd, and yours may be different from mine. —Alessandro Rubini, rubini@linux.it

Blank Screen

Whenever I try to run X Windows, I get a blank screen after some disk action. I've tried the S3 server, an S3 card, the SVGA generic, the VGA generic and the unsupported VGA generic; all give me the same blank screen. I know my settings are right for everything else. I can continue working in other VCs, but they are all blank as well. I know I'm using them, because I get beeps at the right time, etc. I guess it's a problem with my video settings, but I've tried everything and can't fix it. —David Tilleyshort, Slackware 3.4

Odds are there is a mismatch between the frequency settings you have passed in your X configuration and the frequencies supported by your monitor. Make sure you have the correct frequency ranges for your monitor in the XF86Config file; check the documentation that came with your monitor to be sure. If you don't have the documentation, a good generic set of ranges for a low-end multi-frequency SVGA monitor are 31-38 horizontal and 50-90 vertical.

One side note: some “green PC” monitors actually shut down after X starts. There may be parameters which can be passed to your X server to disable this behavior; check the various README files and documentation on XFree86 Org's web site: http://www.xfree86.org/. —Erik Ratcliffe, erik@caldera.com

Establishing an ISP Connection

I can establish an ISP connection. I cannot use a browser, FTP or TELNET. I get an error message like “cannot find URL”. I am using Xisp to connect and have made a symbolic link from /dev/modem to /dev/ttys1. Can you help me? —Ric Stattin, Caldera 1.2

Chances are your name server isn't entered in /etc/resolv.conf. The format of this file is as follows (from comments in resolv.conf generated by LISA):

```
# possible entries are:
# gethostbyname syscall is used to
# search <list_of_domains> Search list for
# hostname lookup.
#
# nameserver <ip_addr> Define which server to
# contact for DNS lookups. If there are
# multiple nameserver lines (Max=3),
# they are queried in the listed order.
```

There is also a “domain” entry (of which you can have only one). So, a sample resolv.conf file might look like this:

```
domain caldera.com
search caldera.com personal.net
nameserver 192.168.1.1
nameserver 192.168.1.126
```

These entries will need to be changed for individual setups, of course. The name server IP address will need to be obtained from your system administrator or ISP. Once you have this information in the /etc/resolv.conf file, you should be able to connect to remote hosts by name. —David M. Brown, david@caldera.com

Mounting ATAPI CD-ROM

When I try to mount my ATAPI CD-ROM (I have two Pioneers as masters on 2-IDE and a Delta as slave on 2-IDE; both drives are recognized by the kernel as hdc and hdd), I get an error message which states, “mount fs: type 9660 not supported by kernel”. I have tried to modify the /etc/fstab with:

```
/dev/hdc /mnt/cdrom iso9660 noauto 0,0
/dev/hdd /mnt/cdrom01 iso9660 noauto 0,0
```

I still get the same error. Can you help me? —Kim Nielsen, Red Hat 5.1

This is happening because your kernel wasn't compiled to support the iso9660 file system, and you don't seem to be able to load the module. Try this first:

```
moremagic:~# cat /proc/filesystems
 ext2
 minix
nodev proc
 vfat
nodev autofs
nodev nfs
        iso9660
```

If you don't see iso9660, you should have an isofs module (unless you recompiled your own kernel). If typing **insmod isofs** and then mounting your CD-ROM works, your problem may be that **kerneld** isn't running. Try:

```
moremagic:~# ps auxww | grep kernel[dD]
root 38 0.0 0.3 756 364 ? S 01:15 0:00 /sbin/kerneld
```

If kerneld is not there, you will need to start it yourself. —Marc Merlin, marc.merlin@magic.metawire.com