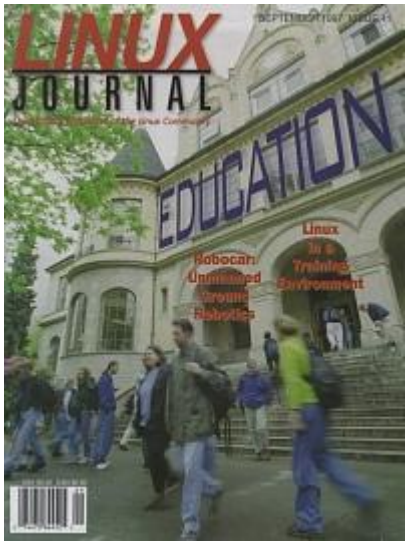
Linux Journal Issue #41/September 1997



Features

RoboCar: Unmanned Ground Robotics *by Kerry Kruempelstaedter*
Students at the University of Colorado at Boulder use Linux on two networked computers which provide the brains for their entry in a robotic vehicle race.

Linux at Holt Public Schools *by Mark Lachniet*
WAN links and linux proxy servers form the basis of a computer network for this Michigan public school system.

A Linux-based Lab for Operating Systems and Network Courses *by Richard Chapman and W.H Carlisle*
Computer science students at Auburn University learn about operating systems and networking using Linux in the computer lab.

News & Articles

Using Linux in a Training Environment *by B. Scott Burkett*
Programming with XForms, Part 3: The Library *by Thor Sigvaldason*
Packet Radio under Linux *by Jeff Tranter*

Reviews

Product Review Empress RDBMD and Just Logic/SQL RDBMS *by Rob Wehrli*
Product Review Megahedron—A 3D Graphics Environment *by Michael J. Hammel*
Product Review SOLID Desktop 2.2 for Linux *by Bradley J. Willson*

Robocar: Unmanned Ground Robotics

Kerry Kruempelstaedter

Issue #41, September 1997

University of Colorado students are using Linux to control their robotic cars in a race with vehicles from around the world.

The Association for Unmanned Vehicle Systems International (AUVSI) sponsors a yearly robotic vehicle contest. Schools from around the world gather to see whose vehicle will navigate an outdoor course the fastest and the farthest. Vehicles have no prior knowledge of the layout of the course.

The course track is defined by white or yellow lines painted on the grass, and the robot must stay within these lines. Obstacles of various sizes, sand traps, deep grades and sharp curves occur along the way to keep things exciting. The difficulty increases as the robot progresses down the course. In addition to navigating the pathways, the robot must carry a 20 pound load and cannot exceed a speed limit of five miles per hour. Each robot has three tries at navigating the course, and the winner is chosen based on the time used and distance traveled. Penalties are assessed for crossing lines and hitting obstacles. So far no team has successfully navigated the full length of the course.

This year's contest was held at Oakland University in Rochester, Michigan on May 31st, June 1st and June 2nd. Rules for the 1997 contest can be found on the Web at http://www.secs.oakland.edu/SECS_prof_orgs/PROF_AUVSI/rules97.html.

Figure 1. Diagram of the Race Course

Robocar is the University of Colorado (CU) at Boulder's entry in this contest. Starting with a child's electric car, our team added basic sensing and control equipment as well as two computers running Linux and software specifically designed for the AUVSI contest. Over the four years CU has entered this contest, Robocar has changed significantly. This article documents the various

systems of Robocar in this year's incarnation, its software architecture and navigation algorithms.

The Mechanics

The basic frame of Robocar is a kid's vehicle, hacked apart—the kind of toy one or two children can sit in and drive. We ripped out the wimpy motors that came with the original toy and replaced them with big, beefy ones with chains and gearing to improve drive power. We substituted computer controls for the steering wheel and pedals and augmented its body structure. Not much remains of the original vehicle except some framework and the outer plastic shell.

An inverted metal “U” was welded to the car body above the former location of the windshield and two cameras were mounted to it. The original windshield was not used because it was too low to the ground and did not provide a large enough field of view. A metal box used for housing various circuits is also attached to the metal “U”.

Batteries fit snugly into the former seating area of the vehicle. Covering the batteries is a wooden board which holds the competition load. Likewise, a wooden platform has been added to the back of the car to carry our main computer, a conventional desktop machine with a big monitor. Additional equipment, including our smaller second computer, is stashed under the hood in place of the original battery.

Overall, the car measures 30 inches wide, 54 inches long and 62 inches high and weighs in excess of 200 pounds. Actually, the car's body has become a huge, heavy hack after three years of adding features. We'll probably rebuild it from scratch next year using lighter materials.

Robocar must withstand a fair amount of stress while traveling across the bumpy course (grass, sand, simulated pavement, wooden ramps). Vibrations need to be dampened to get clean, useful video—try driving down the road while looking through a jiggling video camera some time, and you'll get a fair approximation of Robocar's vision of the world. Besides, we don't want the hard drives bouncing around all over the place; therefore, all four wheels have shocks to help reduce the vibrations, and our mechanically fragile equipment is mounted on foam.

Finally, we replaced the slippery, stiff plastic wheels that came with the car with inflatable rubber tires to improve traction on the grass and the ramps. Robocar plows right through sand traps on these tires. Running with the tires slightly deflated helps absorb shock.

Figure 2. Picture of Robocar practicing. At this time we were using wall power to conserve batteries and a joystick to manually select the motor power. It really is following the lines by itself though. Our practice lines are impossible to miss compared to the more subtle effects of spray paint on grass.

The Electrical System

This year, Robocar has large marine batteries that can easily power it for seven or more hours of operation. The batteries are used in pairs and power all the actuators, sensors and computers. These are deep cycle batteries; that is, they are designed to withstand numerous complete draining and recharging cycles. Each 12 volt battery can source 550 amps. Even though each battery lasts a long time, we keep two spare sets on the sidelines for backup. Unfortunately, the batteries are quite heavy—adding an extra 40 pounds each. The weight trade-off is well worth the power gain, which should enable us to climb the ramp that caused us so much grief in previous years.

Figure 3. A close up of one of the shocks

We have two circuits in the car: a 12 volt circuit and a 24 volt circuit. Power for the noisy drive motors and CAN-AMP servo is provided on a separate circuit from the digital devices. Relays switch the drive motors from forward to reverse as well as cut power to the motors in emergencies. The diagram of the electrical system provides more information.

If Robocar should ever go wildly out of control, a quick slap on one of the two emergency stop buttons (one of which is a remote control) will quickly bring it to a halt by disconnecting the motors from the batteries. Even though the car is moving at a mere 5 miles an hour or less, it can still do a lot of damage to objects and people. I have scars to prove it.

Actuators

Every robot relies on actuators to act upon its world. Robocar has three of these:

1. Steering control is provided via a CAN-AMP. The steering CAN-AMP is one of three nodes on our CAN (Controller Area Network). The other two are an encoder wheel and the CAN-PC controller card. Servo behavior can be completely controlled; for example, we can tell it to turn a certain distance within a certain period of time and to decelerate gently before it gets there. Two years ago, we used one of these to turn a single camera rapidly from side to side without damage, because of the great number and flexibility of the parameters to the CAN servo.

2. Motor control is achieved through pulse width modulation (PWM) from a computer to two DC drive motors. These 24 volt motors are extremely powerful and have great torque. One afternoon, we took turns riding on the car, and the motors easily pulled the car and a heavy (185 pound) human passenger up a steep hill. We generate a PWM signal from two cascading counter/timers that receive the same clock signal. The first is set up to periodically generate a rising edge on its output and determines the frequency of the PWM signal. The period of the signal does not change. The output of the first counter/timer is connected to the gate on the second counter/timer. The second counter/timer determines the duty cycle of the PWM signal. A short count on this timer maps to a longer fraction of the PWM period that is high and, thus, to more power being sent to the motor.
3. Shadow-reducing head lamps are switched with a computer-controlled relay. These lights improve the vision sensors' ability to spot the course boundaries.

Figure 4. A nasty bird nest which also serves as the wiring for Robocar

Sensors

To perceive its environment, Robocar needs sensors. We have given it cameras for detecting the track boundary lines painted in the grass, a scanning sonar for obstacle avoidance and an encoder wheel for speed detection. Robocar has some additional sensors for side projects which are not used during the competition.

Vision is supplied from two standard video cameras fed through two Matrox Meteor frame grabbers. We have two different Matrox cards: the Meteor and the Meteor/RGB. Both can read from multiple cameras and grab high-resolution 24-bit color images. The only difference is that the Meteor/RGB can grab frames from a split-RGB source, whereas the regular Meteor cannot. Even though we could plug two cameras into a single Meteor board, we are using two boards to get 30 frames per second per camera. Matrox's Meteor boards are inexpensive, reliable and well supported.

A single Panasonic sonar sensor mounted on top of a Futaba RC servo acts as an obstacle detection device. It scans the area in front of the car, rotating back and forth to cover a wider area. Using a single sonar has the advantage of removing any possibility of cross-talk and of being able to look in any direction. Using multiple statically-mounted sonar sensors would not give us this much flexibility. The Futaba servo, like the drive motors of the vehicle, is controlled using PWM.

An encoder wheel returns data to a speed sensor indicating how far it has turned. Since we know the diameter of the wheel, we know how far it has turned since last we checked. Thus, this sensor can compute our average speed during that time. The sensor's interface to the encoder wheel is through a CAN-PC board on our main computer. Robocar uses this sensor to ensure that it stays under the 5 MPH speed limit.

In addition to being a competition vehicle, Robocar acts as a test bed for Kevin Gifford's Ph.D. thesis, which is to develop an efficient navigation algorithm for (possibly off-world) autonomous rovers. An additional set of sensors has been added for this option: a GPS sensor and a “map” sensor. Using these, Robocar always knows exactly where it is and where it wishes to go; it can also plan the cheapest way of getting there.

The Trimble Series 4000 uses differential GPS and can make extremely accurate measurements—+ or - 10 centimeters—compared to normal civilian GPS. It comes with a base station, a receiver and radio modems. GPS information is supplied over a serial line.

During Kevin's research, Robocar knows about its environment by using a map sensor in addition to the competition and GPS sensors. The map sensor is basically a topological map of the research field. With this knowledge, Robocar can calculate the most efficient path to a set of destination coordinates.

In addition to the above sensors, we have a joystick for manually driving Robocar to and from the course (or around the test field just for fun). Without this, we would have to push or carry the heavy beast around—something we prefer to avoid. The joystick is plugged into a generic sound card on our main machine.

Computers

Two networked computers provide the brains for Robocar and the control for sensors and actuators. Debian Linux version 1.2 is installed on both these machines.

The first of these, Highlab, is a Pentium 166MHz with 16MB of RAM and a 1GB disk. The three boards in Highlab for sensor and actuator control are:

1. An ML16-P analog and digital I/O card made by Industrial Computer Source. The ML16-P is a low-quality, low-cost real-world interface for the ISA bus. It has sixteen 8-bit ADCs (analog to digital converters), two 8-bit DACs (digital to analog converters), eight digital output lines, eight digital input lines, and three 16-bit counter timers. We use this card for PWM motor control, e-stop, reverse and head-lamp relay toggling.

2. A CAN-PC card made by OmniTech for communicating to their CAN devices (the encoder wheel for speed sensing and the big servo for steering).
3. Two Matrox Meteor cards used for vision.

Highlab makes the high-level decisions and controls all of the actuators. It also performs vision and speed sensing.

Flea, the second of the two computers on Robocar, is a PC/104 stack. The PC/104 is an embeddable implementation of the common PC/AT architecture. It consists of small (90 by 96 mm) cards which stack together. A PC/104 uses ISA compatible hardware, although the connectors and pin-outs are different. Any software that runs on a regular desktop machine will also run on a PC/104. Its greatest advantage over a desktop machine, besides its compact size, is its greatly reduced power consumption. For more information on the PC/104 standard, see http://www.controlled.com/pc104/

Flea consists of several modules: a motherboard (the CoreModule/486-II from Ampro), an IDE floppy controller (the MiniModule/FI from Ampro), a digital I/O card (the Onyx-MM from Diamond Systems) and an Ethernet card (the MiniModule/Ethernet-II from Ampro). It has 16MB of memory and runs with a single 20MB solid-state IDE drive (the SDIBT-20 from Sandisk).

Since Flea has no video card, it uses a serial terminal as its console. We needed to patch the kernel to gain this ability, as it is not part of the normal kernel distribution. The serial console patch can be located at ftp://ftp.cistron.nl/pub/os/linux/kernel/patches /v2.0/linux-2.0.20-serial-cons-kmon.diff

The Onyx-MM features 48 digital I/O lines, 3 16-bit counter/timers, 3 PC/104 bus interrupt lines and an on-board 4MHz clock oscillator. Flea controls the scanning sonar's servo with this card. Sebastian Kuzminsky's Linux driver for this card can be found at ftp://ftp.cs.colorado.edu/users/kuzminsk/

Flea's task is simple; it turns the servo, pings the sonar and listens for the response. When it has a complete sweep of the arc in front of the robot, it processes and sends the information to Highlab.

Software Architecture

This year's software, running under the Linux OS, is significantly improved from last year's, which ran under MS-DOS. Although the MS-DOS system worked fine (we won third, first and fifth place in the previous three years), it was extremely difficult to expand, ugly and monolithic. As soon as Sebastian finished developing Linux drivers for all our unsupported equipment, we completely

removed any and all traces of MS-DOS from our systems and rewrote the code from scratch.

Functionality has been modularized into two types of programs: a single arbitrator which makes the decisions and controls the car, and sensors which provide information about the world to the arbitrator. Sensors are derived from a skeleton sensor and are easily created. You write the code to create a suggestion, to interface to the hardware and to link to the sensor library. The arbitrator and the sensors use a common configuration library which makes it easy to parse configuration information from the command line and configuration files.

Since the sensors and the arbitrator can run on any machine on the Robocar network, it is simple to add and remove computers to and from the system as needed. The arbitrator spawns sensors at startup using **rsh**. A simple command protocol allows communication between the sensors and the arbitrator over the network. The arbitrator can get and set a sensor's configuration, get a single suggestion from a sensor, set a sensor's suggestion rate and kill a sensor. Acknowledgments from the sensors are necessary, since we are using unreliable UDP (User Datagram Protocol) as our networking protocol.

Sensors generate several types of suggestions for the arbitrator: an occupancy grid, the current speed and (for Kevin's research only) a heading. Occupancy grids are just a way of representing world information in a grid format. Our occupancy grids are 6 meters wide and 3 meters high and have ten grid points per meter. The car is centered in the middle at the bottom of the grid. Each point of the grid can be marked with one of three values: good (it is okay for the car to move to that spot), bad (the car should avoid that position) and unknown. Not all sensors provide occupancy grids; those that do are only looking for specific types of “badness”—track boundaries (vision sensors) and obstacles (sonar sensor). In the future, we will probably allow the sensor to use weights of badness instead of a single value, so that the arbitrator can better choose between two “not-so-good” paths. Sensors send suggestions to the arbitrator as fast as they can, at a specified rate or on demand via UDP. These are not acknowledged by the arbitrator and can get dropped if the network gets bogged down. This protects the arbitrator from sensors that send suggestions too fast. Time stamps on the suggestions lets the arbitrator know how old the suggestion is.

The user can configure and debug sensors and the arbitrator from nice menus displayed using curses library routines. The arbitrator itself may wish to configure the sensors; for example, it may wish to alter the suggestion rate for a particular sensor or to change the type of filtering done by a sensor.

After spawning the sensors, the arbitrator waits for each sensor to connect to it and then gathers configuration information from all of the sensors for later use and display. Finally, it falls into a loop. Within the loop, the arbitrator selects from all of the sensor file descriptors and standard input to gather suggestions from the sensors and commands from the user. Using the suggestions, the arbitrator makes a navigation decision and actuates.

Navigation Algorithms

We have several navigation algorithms to choose from and can switch among them on the fly. However, we have found that the simplest and easiest one works best. The Robocar needs to make only local decisions and does not need to keep a map of its environment. It just needs to make quick use of the data provided by its sensors.

Rather than looking at all the sensor information separately, the arbitrator merges the suggestions together into one total suggestion. It then looks at the occupancy grid portion of the total suggestion to find badness in relationship to itself. Badness can be either a painted line or an obstacle—it doesn't really matter to the robot which—and must be avoided. The robot looks left, then right to find the left-most and right-most badness. It then tries to steer between the two. If there is badness only on one side, it tries to give the badness wide clearance—at least half the track.

This is one of many algorithms to which we can switch, but it seems to work well and is fairly straightforward. Kevin, of course, uses different algorithms which take into account current and desired position as well as surrounding terrain. Our simple algorithm works well for the competition.

Conclusion

Working on the Robocar project has been a very rewarding and exciting experience. There is nothing quite so pleasant as watching something you have built and programmed move on its own. Switching to Linux has allowed us to improve our robotics software and to use our favorite development tools. We hope to do well in this year's contest as a result. But even if we do not, we will have a good platform for next year and will have learned a little more about building robots and robot navigation.

Contributors/Contacts

The Robocar Team

Robocar Race Results

Kerry Kruempelstaedter can be reached at kruempel@cs.colorado.edu or at http://ugrad-www.cs.colorado.edu/~kruempel/. Since graduation, she has greatly enjoyed working with robotics and is taking the summer off to work on an autonomous aerial vehicle. She spends too much of her life spelling her name to people over the phone.

Archive Index Issue Table of Contents

Advanced search

Linux at Holt Public Schools

Mark Lachniet

Issue #41, September 1997

How a public school system in Michigan has used WAN links and Linux proxy servers to upgrade its computer network.

Holt Public Schools is located in the mid-Michigan area and is comprised of approximately 5,400 students, ranging from pre-school to 12th grade. We have a wide area network of 10 schools and an administrative office building. Within these schools are a total of about 900 workstations: some 600 Macintosh machines (LC3s and Centris 610s) and around 300 Intel machines. The network, as installed, was comprised of a number of 10MB/s Ethernet LAN segments connected by 56K leased lines. The DSU/CSU units were controlled by Novell 4.x file servers using Newport Systems LAN2LAN cards and routed only IPX traffic. The Novell file servers provided NetWare services to the Intel machines and AppleTalk services to the Macintosh machines.

Since 1993, the computer networking world has been turned upside down, and we recognized the need to update our system. In addition to the need for greater bandwidth for applications, getting Internet access to the classrooms has become a must. Public schools have always been faced with tight financial budgets and implementing a major change in a network is usually quite costly. Because of these factors, our district was forced to examine alternative ways to upgrade our system. Through the combination of two key technologies, spread spectrum wireless WAN links and Linux proxy servers, we were able to provide what our district wanted: a system that is highly functional and low in cost. In fact, based on our financial calculations, the entire system (including the Linux machine hardware) should pay for itself within 5 years, based solely on the recurring costs of six leased lines.

There were two major problems in upgrading our network. The first was that our WAN links were slow and very expensive. Using 56K DSU/CSUs, we were able to pass GroupWise e-mail, do some management and replicate our Novell directory services. At this speed, copying files was slow, and Internet access for

the entire district was unthinkable. To provide for more bandwidth, we replaced our 56K lines with a wireless 4MB/s bridge unit from Pinnacle Communications of Dayton, Ohio. The wireless links can transmit data at a rate of 2MB/s on each channel simultaneously and function as an Ethernet bridge.

An Ethernet bridge, in its simplest form, is a device that will selectively forward or drop a packet based on its destination. The bridge learns which network devices (based on its unique MAC address) are on which network interface and records the information into its internal tables. When a packet reaches the bridge, the bridge looks at its destination. If the packet is destined for the opposite side of the bridge, it is forwarded. If the packet is destined for a network device on the same side of the bridge, it is ignored. In this way, the bridge passes only packets that need to be sent and eliminates unnecessary traffic between segments. The wireless bridge product we use performs this exact function between an Ethernet segment and a wireless “virtual” network. This makes network configuration very simple and efficient.

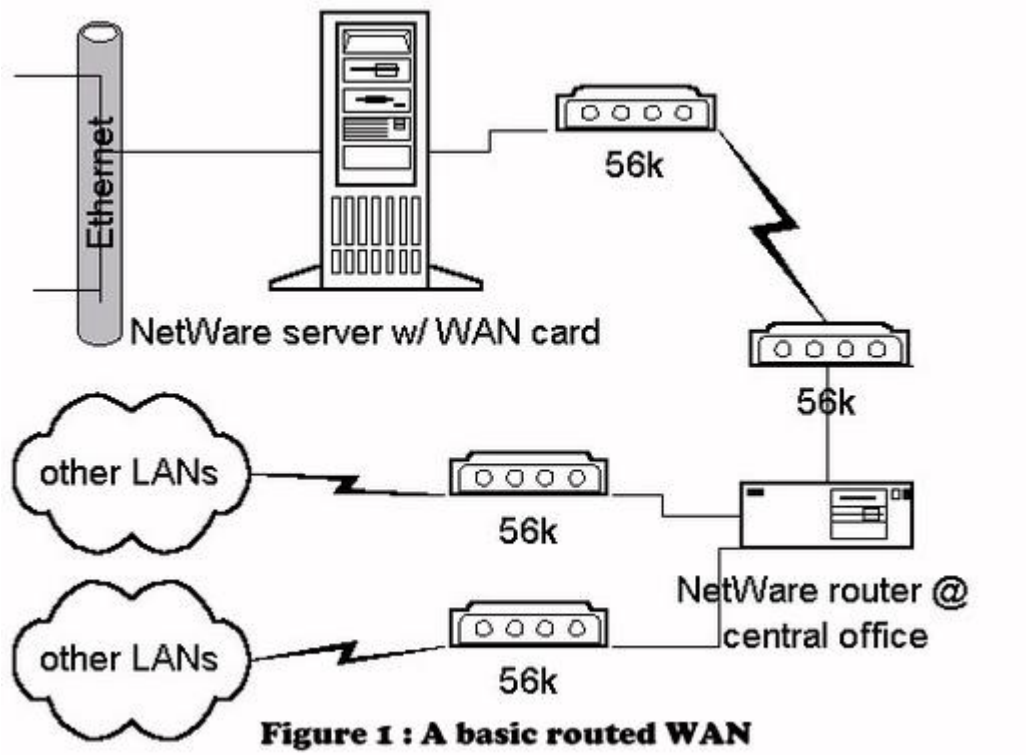


Figure 1 : A basic routed WAN

Originally, we used a routed 56K solution such as that shown in Figure 1. This worked relatively well for our Novell network. However, it required that all of the traffic be routed. This is fine for NetWare, which uses protocols such as RIP to automatically configure routes. However, in order to pass TCP/IP data, it would be necessary to break our class C range of IP addresses into a number of smaller subnets. This would result in a net loss of available IP addresses and add to the bandwidth problem.

With the wireless links, we were able to set up a much more flexible network. Since the links can be configured as either point-to-point or multi-point, it was

possible to create a single, virtual-bridged radio network comprised of numerous locations. At the education center, we have OMNI-directional antennas which are configured as repeaters. Because the remote locations use directional links and point directly at the education center, they cannot communicate directly with one another. To alleviate this problem, the education-center bridge is configured to forward all traffic not intended for its own Ethernet LAN segment back out to the wireless network. Thus, while one elementary school cannot communicate with another elementary school directly, they can communicate by making an extra hop through the OMNI. This happens transparently in the hardware and is unseen by the network devices. With the exception of a single remote school (Dimondale), we were able to connect every location into a single wireless network. We used a repeater at the west campus area to connect this auxiliary school to the larger network. At this location, there are essentially two different wireless links plugged into the same hub. Although the link to Dimondale is actually a totally different wireless network, it appears, logically, to be part of the larger wireless network. The complete physical wireless network looks something like the picture in Figure 2.

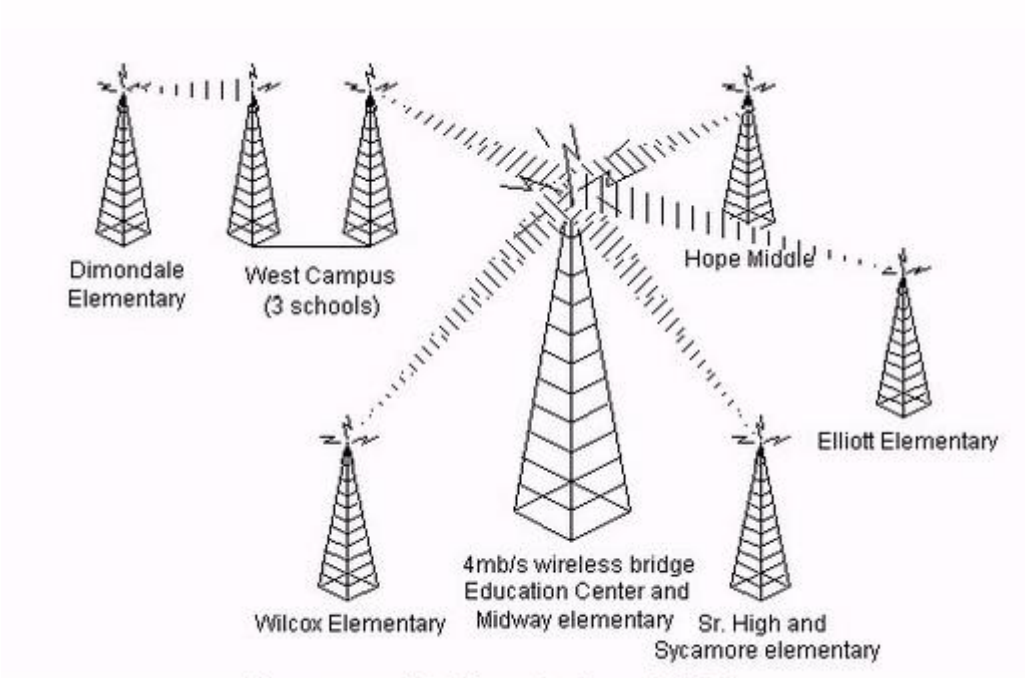


Figure 2 : Holt's wireless WAN

With the bridging topology, we were able to maintain our Novell communications while expanding our TCP/IP functionality. For our TCP/IP services, we deployed a number of Linux proxy servers. These Linux servers are Pentium computers, ranging from Pentium 90s to Pentium 150s. They have between 32 and 64MB of parity RAM and IDE hard disks from 850MB to 2.1GB. They each have two D-Link NE2000 compatible Ethernet cards. The machines have a minimal Slackware 3.2 distribution installed, are configured as IP Masquerading firewalls and act as Internet gateways for our remote LAN segments. Also running on the servers is the Squid Internet Object Cache software, which allows us to cache HTTP, FTP, GOPHER and WAIS data on the

server. Most of the other Linux software, such as login shells, FTP services, etc. have been disabled or restricted to a single management machine.

Between IP Masquerading and the Squid Object Cache, we were able to provide the necessary Internet services. With masquerading, we gave access to our 900 or so clients with only 7 true IP addresses. In addition, we can configure the firewall to allow different types of access to different workstations. For example, we might wish to configure the firewall in such a way as to restrict a computer lab while allowing a teacher station access. Also, because of the nature of the firewall, our clients are more or less unreachable by the outside world, thereby conferring a certain amount of security. Using the standard **ipfwadm** program, there are a number of possible configuration options.

Overall, the speed of the wireless links has been quite good. When the network first went up, we began gathering information about the speed and reliability of the links. To do this, a script was set up which runs a program called **tcpspray** to transfer 100KB of data to each location and measure the amount of time it takes to get there. The script runs constantly on a management station and tests each of the links every 15 minutes. Below is the actual output of one of our wireless links—in this case, between the education center and the senior high school:

```
Tue May 27 18:19:35 EDT 1997 Sycamore/HS: Transmitted
102400 bytes in 0.551536 seconds (181.312 kbytes/s)
```

Running this same test to a machine connected via PPP on an external USR 28.8k, we had the following results:

```
Tue May 27 18:25:43 EDT 1997 PPP: Transmitted
102400 bytes in 47.041576 seconds (2.126 kbytes/s)
```

Admittedly, that PPP link should be running a little bit faster. I had expected to see throughput more along the lines of 3K, or possibly 4KB/s. Lastly, to compare it to another popular networking option, take a look at the throughput attained by a 10MB/s LanCity cable modem which I have connected to my personal Internet host:

```
Mon May 26 18:28:13 EDT 1997 Cable Modem: Transmitted
102400 bytes in 0.294357 seconds (339.724 kbytes/s)
```

Bear in mind that these figures don't give you the whole picture. For example, the speed of the throughput varies from moment to moment by as much as 50%. All it takes is a split-second delay in the network to give a very poor reading. The readings I have provided represent an average throughput. Also, the speed of the computers sending and receiving the information makes quite a difference. All the machines used for the above testing are running Linux, but if they were not, the speed of the TCP/IP stack would also be a factor. In addition, it's important to note that certain services have more data overhead

than others. Thus, performance might vary depending upon the service you are using. FTP, for example, runs at roughly the same speed as a **tcpspray** test.

In addition to speed, controlling Internet access is important. As a public school district, we need to have a certain amount of accountability as to what our students do and see on the Internet. To this end, we have made the use of the Squid Cache mandatory, allowing us to monitor the kinds of documents accessed, as well as require a valid user name and password to access the cache. By filtering all traffic for port 80 at the firewall, client workstations must use the cache to get WWW documents. Squid allows for the use of *htpasswd* style authentication, exactly like web server packages such as Apache. Using this authentication method, we can manage user access to the cache and to the World Wide Web. In addition, Squid allows us to configure access control lists, which will disallow certain "known naughty" sites that might endanger the innocence of our students.

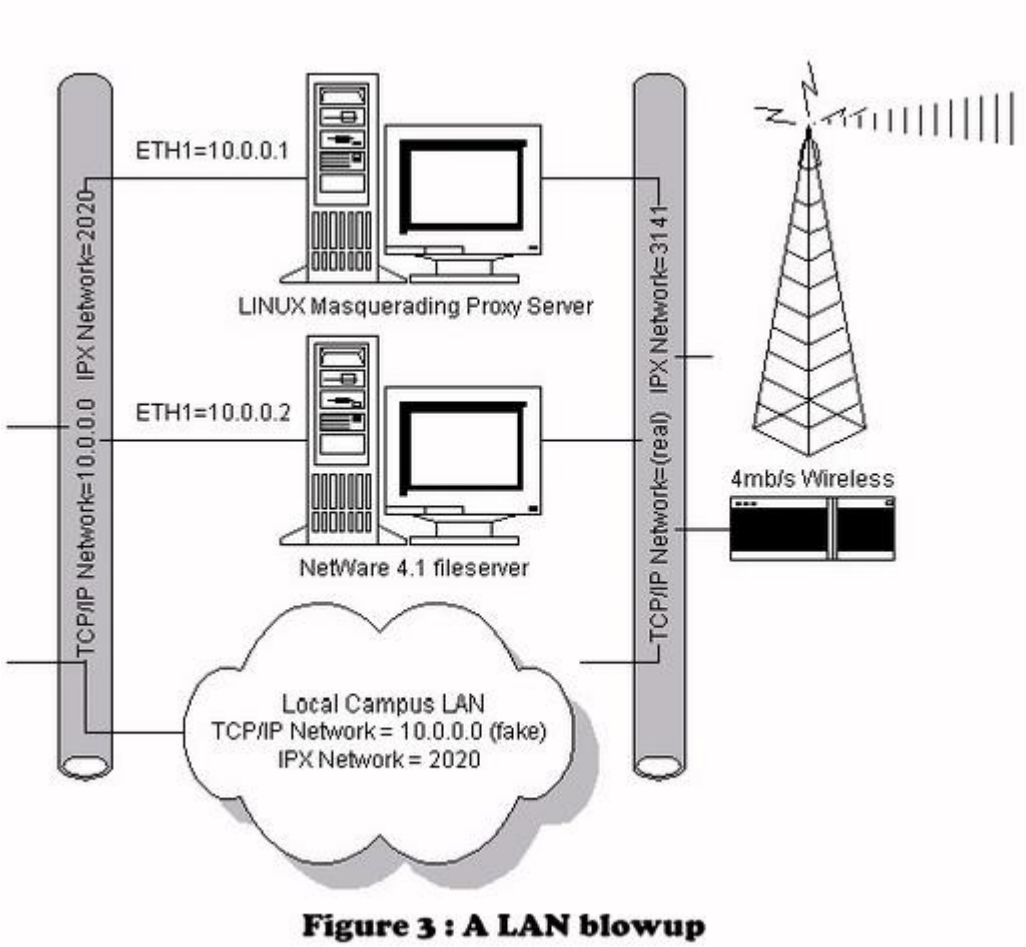


Figure 3 : A LAN blowup

At each of the individual LAN segments, we have a configuration similar to the picture in Figure 3. The wireless link is plugged into a small hub, which is also connected to the NetWare server and Linux box. The NetWare server is responsible for routing the IPX/SPX traffic and the Linux box is responsible for the TCP/IP traffic. All of the client workstations are configured for the 10.0.0.0 network and have their Linux box set as the default gateway. When a client

sends TCP/IP traffic, it goes through the Linux box, out through the wireless link, to the education center and eventually out to the Internet.

On the Novell side, configuration is quite simple. We simply left the original Ethernet card at its original settings and configured a second Ethernet card for the wireless LAN. Naturally, I had to configure this second card with the network number 3141, so I could call it the “Pi in the Sky”. We must all seek humor where we can.

The proxy servers have been working quite well. Our first Linux proxy server, which was based on kernel version 2.0.18, has run for months and never crashed. Even with the ever-demanding (and ever-popular) PointCast traffic, it has performed wonderfully. With Squid, all documents that pass through the proxy are cached, allowing subsequent requests to come from the proxy server at near-Ethernet speeds. This reduces traffic across our Internet connection, as well as across the Internet in general. In a school situation, this works very well. For example, when a teacher wants to visit a certain web site with the whole computer lab, he simply views the pages the night before. When the class comes in the next day, the documents are served very quickly, making it possible for a whole class to browse the site at Ethernet speed. With the combination of helpful utilities such as **wget**, teachers can recursively cache a whole site with a simple shell command.

Using Linux has allowed us to put together a greatly improved network and provide Internet access to all our workstations using limited resources. We have increased our WAN bandwidth from a painfully slow 56K to a respectable 4MB/s. With the help of the Squid Internet Object Cache, we have become responsible Internet citizens and reduced unnecessary net traffic. Now, we can even call all of our e-mail airmail. None of this would have been possible without the effort of the hundreds of people who contribute to Linux every day. In education, in particular, Linux fits perfectly—it's cheap, it' s flexible and it's powerful.

Mark Lachniet is a Network Systems Specialist for Holt Public Schools. Mark's hobbies include disc golfing, “nerdin” and home brewing beer. Mark can be contacted at lachniet@pilot.msu.edu. Mark maintains a small home page which includes a tutorial on IP Masquerading, a HOWTO-in-progress on creating a proxy server like the ones described above and other miscellaneous stuff. His home page is accessible at http://scnc.holt.k12.mi.us/~lachniet/.

A Linux-based Lab for Operating Systems and Network Courses

Richard Chapman

W. Homer Carlisle

Issue #41, September 1997

Auburn University uses Linux as the OS for their computer labs dedicated to teaching students about operating systems and networking.

This article describes our experiences installing and using a teaching laboratory based on the Linux operating system. The lab is a platform for undergraduate operating systems and networking education in the Department of Computer Science and Engineering at Auburn University. We have deliberately made the software and hardware environments of the lab quite heterogeneous, but Linux has always been the workhorse of the lab. Linux was chosen primarily due to the wide range of hardware supported, the sophistication of its kernel and the availability of source code and documentation. We believe that “hands-on” experience is an essential component of computer science education and that most current curricula rely far too heavily on simulation when teaching systems issues.

Teaching Systems Programming

Our primary motivation for building this laboratory was to not shortchange our students when it comes to practical experience with systems programming. The shortcomings we most desired to avoid are:

- Out-of-date technology—A good systems programming instructor must be familiar with the state of the art, and more importantly, the state of practice. A competent instructor should be aware of such things as memory speeds and sizes, disk drive performance, network standards and CPU performance. These things change rapidly, and far too often we throw up our hands and say “I'll teach the fundamental concepts, but not

dirty my hands with implementation details." Such thinking is fundamentally flawed when applied to systems programming, since what works and what does not work is fundamentally based on the economics and technology that makes some solutions viable and others not.

- Too much reliance on simulation—Systems programming is inherently a messy business. Machines can crash; device drivers can hang the system or even the network; hardware can be damaged by poor programming or incorrect wiring. There is a risk that students will use the systems lab to crack other systems. Poorly written network software can result in inadvertent denial of service to other legitimate users of the network.
- All these things make a dedicated systems laboratory very difficult to accommodate in today's typical university computing environment of networked Windows PCs and commercial Unix workstations. Systems administrators don't want to put at risk a part of their network which is, almost by definition, broken most of the time.
- Thus, systems programming courses often rely exclusively on the use of simulators to provide students with some experience in systems programming. The problem with this approach is that the real world is not running on a simulator. Simulation adds an extra layer of complexity that must be grasped by the novice, and also insulates the novice from the fact that development tools may be primitive or nonexistent and the designer is often responsible for building the design tools as well. Students need to learn to build their own tools, to leverage the capabilities they have in order to achieve the ends they need when developing new systems.
- Aversion to team exercises—We in the computer science and engineering department at Auburn often ask our industrial contacts, "What should we be teaching our students to prepare them for work at your firm?" Inevitably, we are told that the important skills include teamwork and experience with large systems. Because of the difficulties inherent in assessing team performance and in managing the interpersonal issues involved with team-based projects, most university exercises tend to be individual activities based on small, or even toy systems. Perhaps the underlying obstacle may be an unwillingness to depart from the traditional lecture-and-examination method of teaching. Professors have a term that is generally used to refer to team work—cheating.
- Security concerns—It is a fact of life that not all university students can be trusted in the same ways that employees are trusted. As mentioned above, some protections must be set up so that hacking does not interfere with the work of other students or with that of university staff. Further, physical precautions must be taken against the theft of equipment available for public use. Unfortunately, these concerns seem directly aimed at frustrating any attempt to provide students with "real" systems programming experience, for which they may need root

privileges, and for which they will often need access to the inside of the machine.

Figure 1.

A dedicated laboratory for systems programming courses, running Linux on commodity hardware, isolated to some extent from the main campus network, seemed to us to be a good way around most of these shortcomings.

Lab Configuration

The lab began in the late summer of 1995 with six Pentium 90 machines running Red Hat Linux version 1.0. We made an effort to buy only hardware that supported Linux: IDE disk drives and CD-ROM, SCSI tape backup and file servers, ATI Mach 32 video cards, SVGA monitors. Hardware incompatibilities caused several delays, as we attempted to show that Linux was competitive with commercial Unix as a platform for instructional computing. These delays seem to be a thing of the past, both due to our increasing familiarity with Linux and to the non-stop efforts of the Linux development community.

The “PC lab running Unix” initially was regarded rather like the dancing bear: People were impressed not that it worked well, but that it worked at all. Faculty were amazed that we could run essentially the same set of tools on $2000 boxes as ran on their $4000-$8000 Unix workstations. Benchmarking and practical experience on tasks ranging from document preparation to compiler writing soon revealed that performance on these machines was comparable to that achieved with workstations. Further, the fact that all OS source code was available, and that the developers could be contacted directly (and generally provided very helpful advice via e-mail) were big pluses. The machines went through a shakedown period in the fall of 1995, in which any interested student or faculty was granted access, but the machines were not used as part of any formal course work.

Figure 2

Expansion

During the shakedown period, we attempted to attach virtually any peripheral we could find to our Linux system and attempted to install Linux on a variety of processors.

The initial six machine configuration was used to support a systems administration course during the spring quarter of 1996, with teams of about six students assigned to each machine. Students installed Linux, configured networking, mounted file systems over the network, installed devices,

developed device drivers and made minor kernel modifications. It was becoming clear that with more machines, the lab could be a platform for most, if not all, of our systems courses.

At about the same time, we scavenged a large number of 386 and 486-based machines that were being surplussed as a large college of engineering laboratory was upgraded, and while we found many broken computers, we were able to build somewhere between 10 and 20 working systems, all capable of running Linux. We wanted an environment that was varied and in which hardware was plentiful (if not always state-of-the-art). These older machines provided a platform that could be freely used without worry about damage by any student interested in hacking. One such project included interfacing a color Connectix QuickCam which exported pictures of the laboratory to the World Wide Web and to a file server, thus providing both security and convenience at a low cost.

Figure 3

Based on our preliminary results, we submitted a proposal to the National Science Foundation (NSF) Instructional Laboratory Improvement program to expand this lab, and to use it as our main platform for undergraduate education in operating systems and networks. The NSF agreed to supply $44,512 to purchase equipment, if Auburn University would provide a matching $44,512. The total $89,024 was allocated toward purchase of the following items:

- Sixteen additional Intel-based computers
- Fast Ethernet cards, cables and hubs
- Network analyzers
- Multimedia equipment
- Printers

We are currently in the process of acquiring this equipment and developing a curriculum for introductory operating systems and networks courses based on the Linux kernel.

For the introductory operating systems course, we currently use the Nachos instructional operating system, developed at the University of California at Berkeley. We start the students out with a very limited-capability kernel and require them to extend the scheduling, file system, process management and networking in various ways. This OS runs on a MIPS R2/3000 processor simulator, running on top of Solaris on a SPARCstation. We hope to remove most of these software layers by using Linux. We plan to augment the use of our traditional textbook, *Operating Systems Concepts* by Galvin and

Silberschatz, with a Linux-specific text, such as *Linux Kernel Internals*, by Beck, et al.

Figure 4

Systems Administration Course

A Unix systems administration course is currently being taught using the laboratory. On the second class day students were required to choose a slot in a table having rows reflecting tasks of a system administrator and columns reflecting the hardware platforms available in the laboratory. The rows were called “interest groups” and the columns were called “teams”. The interest groups of the table were:

- Hardware group—responsible for hardware installation, maintenance, upgrades.
- Software group—responsible for software installation, maintenance and upgrade of system software.
- Network group—concentrating on networking hardware and software.
- Backup and security group—responsible for prevention and monitoring of security for the laboratory.
- Documentation group—responsible for dealing with the maintenance of appropriate documentation of a system. The person choosing the documentation group slot for a team was also given the leadership roll for their team.

The above-mentioned Pentium processor systems and two older SPARC processors were the hardware platforms offered to the student teams. Additionally, each team was allowed to choose an operating system to manage on their machine subject to license availability. For the Pentium processors the students were offered Solaris for the 386 or Linux. Solaris or SPARC-Linux was available for the Sun systems. One team expressed considerable interest in integration of Windows NT into the laboratory, and since there was one license available for NT, it was agreed that this team could support NT, so long as class assignments could be performed on their system.

Figure 5

Class assignments were of three forms: user assignments, team assignments and interest group assignments. Because each team was to provide accounts on another team's platform, each team became a user base for another team. Thus, individual user assignments became team assignments. For example, a user assignment to enter an HTML page containing a Java applet became a team assignment to install a web server and a Java compiler on their system.

Interest group assignments were those that altered or reconfigured the basic capabilities of the laboratory. For example, creating two subnets in the laboratory, creating a backup system for the laboratory machines, monitoring and securing the laboratory were duties in which the interest groups joined together to accomplish the assigned task.

Finally, to address distributed system management issues, all teams were assigned the responsibility of installing, providing and managing file and print services for the laboratory. An Iomega Jaz drive was configured as the boot disk for the machine in the laboratory that was to be the NFS and print server. This machine was also configured with an additional two SCSI hard disks, a tape drive and a printer. Each team checked out a Jaz disk and built an operating system on its disk. The only team that had a problem using the removable disks was the Windows NT team, which discovered that (by design) a Microsoft Windows application cannot have page files that reside on a removable drive.

Initially a working system was provided for the file server, but occasionally a team's Jaz disk would become the working system for the laboratory. If problems were discovered, the initial system could easily restore needed services, and the team could be given the responsibility of repairing the problem without disrupting services. Maximum chaos could easily be achieved by assigning problems such as the rebuilding of the kernel on the Java disk system.

Security Concerns

Due to concerns by Auburn's Engineering Network Services group regarding students having root access to machines connected to the campus network, the Linux machines were isolated on a subnet, connected to the main college of engineering network through a secure router which knows the address of the machine connected to each port, and routes packets only to the machine designated to receive them. We physically secured the machines by locking the room in which they were kept whenever no paid employee was in the department (not necessarily in the same room), and we routed a fiber optic cable connected to an alarm box through all the machine cases. We soon found this cable frustrating, because it required the assistance of the department's single systems administrator any time a hardware change was necessary (this occurred several times a day during the first months). In spite of these precautions, we lost the motherboards and disk drives of two machines to theft during that quarter. We had failed to realize that the sort of commodity hardware used in this lab is a much more attractive target for theft than the engineering workstations, minicomputers and parallel machines we have traditionally used for academic research and instruction. Further, the constant need to access the hardware inside the PC cases meant that more opportunities existed for the alarms to be disconnected. There seems to be a

fundamental tension in a systems lab between making the lab a good development environment and making it secure against a dishonest student—one simply cannot lock down every cable, nut and bolt if any work is to get done. On the other hand, insurance policies are difficult to obtain without evidence of adequate security.

Our current security approach is multifold:

- Fiber optic cables run through the machine cases
- A lock on each computer's case—keys are available any time during business hours
- A digital camera, connected to a 486 machine, used to snap pictures, send them to a file server in a very secure room and serve the latest one to the lab's web page at http://dn102af.cse.eng.auburn.edu/~root/labimage.html
- Locking the lab when none of the departmental staff is around

Any individual measure can certainly be circumvented, and of course, a dedicated effort can defeat any security system. So far this system seems to provide a credible theft deterrent without interfering too heavily with productivity.

Lessons Learned

For many of our students, their experience with our lab turns out to be the first time they have academically dealt with the inside of a computer. Thus, we find ourselves teaching a way of thinking as much as the particular techniques related to the subject matter. The commonplace tasks of debugging a system, such as isolating a problem to hardware configuration, BIOS settings, OS kernel or application, are new territory for beginning students. This lab gives them the chance to gain confidence in a realistic setting.

One criticism that students have made of our approach is, “The real world mainly runs Microsoft Windows—why aren't you teaching us about that?” While their claim is indeed valid when one looks at market share and while any computer scientist who claims to be well-rounded must accommodate the demands of the market, there are significant obstacles to providing the kind of experience we provide using commercial software. The two largest obstacles with commercial operating systems are the lack of source code and the lack of easily available technical support from the software developers. A student who wants to explore the performance differences between several different page-replacement strategies in the file system can do so more easily with Linux and a PC than with any other OS and computer of which we are aware.

Another lesson we learned is that with today's large disk drives and with the use of removable cartridge hard drives, it is possible for a number of operating systems to (more or less) coexist on the same machine and for a wide variety of platforms to communicate over a local network. We think this has provided our students with a taste of the complexity of real-world systems administration (though they will not truly know what that is like unless we make them wear beepers and page them at all hours).

References

On balance, we find this Linux-based approach an improvement over our previous methods and plan to continue it. We've had very positive preliminary feedback from the students who have used the lab and are looking forward to hearing whether it helps our graduates in the “real world”.

Richard Chapman is a faculty member in the Department of Computer Science and Engineering at Auburn University. His other interests include hardware/software co-design, formal methods and the history of computing. He has been involved with Linux since the release of Red Hat v1.0. He used to restore early 1970's minicomputers in his spare time—now he wishes he had spare time. He receives e-mail at chapman@eng.auburn.edu.

W.H. Carlisle is an associate professor of computer science and engineering at Auburn University. He received his BS, MA and PhD degrees from Emory University. His research interests are in languages and environments for system software design and testing. He is a member of the ACM and the IEEE Computer Society.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Using Linux in a Training Environment

B. Scott Burkett

Issue #41, September 1997

One company's experience using Linux as the operating system of choice for their training classes.

The proliferation of the Linux operating system has done much to stimulate the interests and activities of hobbyists around the globe. Many developers and users are taking advantage of this freely available, 32-bit wonder, as a viable home alternative to a commercial Unix platform. With a multitude of Linux enthusiasts active in key positions the industry, it was only a matter of time before Linux also became a presented solution in the workplace. Even though Linux sports some of the more advanced and innovative mechanisms available, only recently have many organizations begun to accept Linux as a solution for production projects.

My employer, Decision Consultants, Incorporated (DCI), is one of the nation's largest software services consulting firms. I am a technical instructor in our Training Division, which is responsible for creating, packaging and implementing technical training solutions to both our consultant base and external clients. A large portion of this training revolves around Unix-based client/server development. We needed a versatile, flexible training room, capable of facilitating our entire repertoire of Unix-based courses, from Unix Essentials to X/Motif Development. This formula, once successfully implemented, would need to be “cloned” for our other branch offices around the country. In this article, we will examine certain portions of that solution, as well as present an action plan for implementing your own Linux solutions.

Overview of Requirements

The first step in any major endeavor is to formulate a plan of action which will ultimately lead you to your goal. Our plan of action first involved the creation of

a list of all the major lab-based, Unix-related courses which would need to be delivered in the training facility. They included:

- Unix Essentials
- Advanced Unix/Shell Programming
- C Programming
- X Windows and Motif Development
- Unix Internals/Systems Programming
- Networking with TCP/IP

After the list was created, it was analyzed further to create a list of the tools and host-level resources which would be needed. Of course, all of the classes would require an operating system, a properly configured TCP/IP setup to provide student connectivity and a working Unix shell. A stable C compiler would fit in nicely with the C Programming course, and a feature-packed X server would be needed for the X Windows development course. Another concern was Motif which is available in many forms these days.

Once this list was created, a platform had to be chosen. Given pricing structures of major vendors, such as Sun, HP, DEC, etc., Linux was the clear winner. We could spend several thousand dollars for the latest and greatest version of one of the popular vendor offerings or a mere twenty bucks for a recent Linux distribution on CD-ROM. The choice was not a difficult one to make.

However, the choice of Linux was not based on price alone. As you are probably aware, all of our requirements (with the exception of Motif) have been readily available on most Linux distribution CD-ROMs for quite some time. A number of other nice tools, such as TCP/IP, NFS and a full suite of developer tools, would have cost extra with most commercial vendors. Several inexpensive, Linux-specific versions of Motif are available, such as SWiM, Metro Motif, Mootif and Moteeth.

The total cost for the host-side software load totaled somewhere in the neighborhood of 120 dollars (20 for the Linux CD and around 100 dollars for the X11R6 version of SWiM for Linux). That cost is roughly equivalent to the sales tax you would incur when purchasing a major vendor's bundle.

Implementation

A typical DCI training room consists of eight student workstations connected to a single host machine on an isolated Ethernet bus. The host machine should have the capability to handle at least eight TELNET sessions or, in the case of

our more advanced courses, up to eight X Window sessions. The following is a breakdown of a typical DCI training host:

- 90MHz or better Pentium processor
- 16 to 32MB of RAM
- Adaptec 1542CF SCSI controller (ISA-based card)
- 1GB (or better) SCSI drive
- Standard issue SCSI CD-ROM unit
- Streaming SCSI DAT drive
- Generic NE-2000 compatible network interface card (NIC)
- Number 9 Trio64 (S3 chip set) video card (2MB RAM on board)
- Low-end, LaserJet-compatible printer

The Processor

Our original prototype training room uses a stock 60MHz Pentium processor in its host machine, although newer training rooms are coming on-line with Pentium 90MHz processors or better. There is a huge difference in processing speed between the older Pentium 60 models and the newer Pentium 90 units, although I have had great success with both systems. If you are still hesitant about obtaining a Pentium machine, a 486DX/100 unit will provide comparable performance.

Memory

For most training classes including X Windows and Motif, 16MB is adequate. Of course, greater performance can be obtained by simply upgrading to 32MB of RAM on the host. I recommend getting the full 32MB of RAM initially, rather than purchasing it later. While our MIS department has been quite accommodating to our hardware requests, your organization may not be as generous. If you have corporate red tape to cut through, request the 32MB up front.

The SCSI Controller

We currently use the Adaptec 1542CF SCSI controllers. These are ISA-based cards which have been stable under Linux for quite some time. I have experimented with the Adaptec 2940 PCI-based controller, but it was a bit too squirrelly for my tastes. Even though the 1542 units are 16-bit ISA cards, my aim was stability first and foremost. A few other cards which I can personally attest to are the Future Domain 1680 series and the older Always IN-2000 cards.

The Hard Disk

Our first training room used an older 500MB IDE drive. While it served admirably and reliably, it also reached maximum capacity in a hurry. For a full install of Linux, complete with XFree86, I allow a liberal 200MB or so. However, some other storage requirements must be taken into consideration during the planning phase. For instance:

- Motif—With the newer X11R6 distributions of SWiM, roughly 30MB of storage is needed for a full install from CD.
- Student lab work—Plenty of storage must be set aside for student lab work. Some courses, such as the Shell Programming course, don't require much storage for student lab work. Other courses, like our X/Motif Development course, require quite a bit. For 8 students, I recommend having around 20MB or so available per student for their course work.
- Linux kernels—If you plan on experimenting with newer revisions of the Linux kernel, plan on having a lot of extra room. I recommend having 20MB or so per revision.
- Temporary storage—Plan on setting aside a liberal amount of storage for temporary files (i.e., the /tmp directory). In fact, I recommend that you make this directory a separate file system altogether. I like to have 100-200MB available for a typical temporary storage area.
- WWW storage—We run an internal training Web, complete with on-line prep tests for our students. I must point out that even the smallest working Web requires a good bit of storage. We currently have around 20MB or so of web information on-line (including the web server software and our image library).
- Working storage—Of course, we need plenty of room to sock away on-line course materials (completed solutions to lab work, shell scripts, etc). In addition, our instructors do quite a bit of development and experimentation as well, so that must be taken into account as well. A few hundred megabytes will work nicely.

The CD-ROM Unit

Of course, any good Linux system needs a CD-ROM unit attached. With most software packages shipping on CD-ROM these days (including Linux), it pays to have one of these drives in place. Should disaster strike, it's much easier to reload the base operating system from CD-ROM, rather than a tape backup unit. I have had great success with several models from NEC, Sony and Sanyo. Try to stay away from proprietary SCSI interfaces, such as come with some Compaq CD-ROM drives. That old single-spin, wonder unit in the attic may make a perfect candidate for this job, since it won't be used all the time.

Streaming SCSI DAT Drive

These wonderful devices make perfect solutions for backups. These drives are so fast and quiet that I have actually performed system backups while a class was in session. Any major brand should work nicely, although I can personally attest to the 2GB and 4GB models from Colorado. Even if you have to perform backups on an older 120/250MB Colorado Jumbo, the issue of system and working backups should be addressed swiftly and immediately.

The Network Interface Card

For connectivity, a generic NE-2000 compatible card works rather nicely. I have had good experiences with 3COM 3C509 cards, as well as the Intel Etherexpress cards. If you are setting up a full network, be sure to purchase network cards which match any existing or planned wall connections. Don't run out and save a bundle on a rack of AUI-based cards, if you have twisted pair connections in the wall already.

The Video Card

We use and recommend the Trio64, S3-based cards from Number 9. These cards have proven to be quite reliable and versatile under X Windows. This is a tricky area, due to the fact that XFree86 only supports cards with certain chip sets. Other good choices include cards sporting a Tseng-4000-based chip set.

The Printer

Our students are not in the graphic design business, nor are we. A low-end, LaserJet-compatible unit works nicely for source code printing and other small jobs. In fact, even a low-end printer can generally support postscript or PCL raw formats, so working with programs like ghostscript and TeX can be facilitated easily. I think our printers are coming on-line at around $400 per unit—not bad.

Back to Business

While installing, configuring and maintaining our Linux host machines can be somewhat time consuming, the same procedures, when performed on our student workstations, take considerably less time. Depending on which training branch you visit, our student workstations range from older 486/33 machines to newer Pentium 90 desktop models. For a simple TELNET connection, even older XT/AT or 286 machine is capable of running NCSA's freely available implementation of TELNET.

For X Windows and Motif development, a more robust platform is required. Most of our workstations have around 500MB of storage and 8 to 16MB of RAM. DCI is an Authorized Microsoft Technical Education Center (ATEC). As

such, we also instruct a number of non-Unix related courses, such as Windows NT development. These courses require considerably more resources on the workstation side. The only common denominator is really the network cards used in the workstations, which are also of the NE-2000 variety.

As far as workstation tools go, we use the following software packages on each of our workstations:

- NCSA's TELNET Package has truly been a gift from the heavens. It has performed reliably over a sustained time period and is quite configurable on the workstation side. With it, our students are able to maintain multiple TELNET sessions, as well as the occasional FTP to the host to upload their lab work.
- X/Appeal from Xtreme S.A.S. (Italy) is a remarkable, surprisingly inexpensive X Server for DOS. It supports a number of different video chip sets, as well network configurations. A 30 day trial version of X/Appeal can be obtained at ftp://oak.oakland.edu/.
- Microsoft Windows for Workgroups v3.11? In a Unix-based training room? Surprised? Not at all. One of the nice things about our setup, is that we use the freely available Samba package to allow Linux to provide shared directory and printer services to our DOS/Windows based workstations. The bulky Windows products used in some of our other courses (PowerBuilder, Visual Basic, Visual C++, etc.) can be installed directly onto the Linux host, freeing up valuable disk storage on the workstations. In fact, we even share the CD-ROM which is installed on the host machine. The student workstations can then access the CD-ROM at any time.

Benefits/Drawbacks

While there are a number of advantages to using Linux as a training solution, a number of drawbacks also manifest themselves over time. Do the benefits outweigh the drawbacks? I'll let you be the judge.

The Price of the Operating System

Free. End of story. As mentioned earlier, the price of a stable Linux distribution on CD-ROM is exponentially cheaper than obtaining a commercial solution, such as SCO or Unixware. In fact, with the extra cash you have left over, you can afford to subscribe to Infomagic's quarterly Developer's Resource 4 CD set for the rest of your life.

The Price of the Hardware

With PC-based hardware prices falling, you can pick up an adequate host machine for under $4,000. Compare that to the 5-digit price of a proprietary architecture, such as an IBM RISC machine or a Sun SPARC.

The Price of Tools and Add on Packages

As mentioned earlier, most, if not all, Linux distributions ship with a multitude of packages which would cost you extra from some commercial vendors. A third party Motif derivative for Linux runs far less than the asking price from OSF. In fact, one of the reasons that I became involved with Linux was the steep-pricing structure issued by SCO. I am a former employee of a SCO VAR, reseller and software development house. I decided that I would purchase SCO for myself and run it at home on one of my spare machines. I laid out $1,500 just for the base operating system, only to discover that to add TCP/IP and the Developer Kit another $1,500 would be in order—not for me.

Lack of Technical Support

This is truly the most pressing battle you may need to fight. Since there is no central technical support group for Linux, internal staff are responsible for all maintenance and support of the system. If you don't have a true Linux fanatic around or someone, who plans on becoming one in a hurry, you might be better off with a commercial solution. We have two Linux mongers on our instructor staff, with another dozen or so in our local consultant base—works out rather nicely for us.

Lack of Commercial Solutions

Up until now, most commercial software developers and vendors shied away from marketing Linux native tools. However, a new trend is coming into play. Thanks to some key players in the industry (Caldera, WordPerfect, etc.), more and more tools are becoming available for Linux proper. I expect this trend to continue, as more and more Linux machines appear in the workplace. In addition to the Linux native packages which are becoming available, another option exists. Under the freely available iBCS2 emulator, binaries for other iBCS-supported platforms can be utilized under Linux. In fact, we have had great success running the SCO versions of many packages under Linux, including WordPerfect/X and Oracle 7. While a further discussion of iBCS2 is an entire series of articles in itself, it is something you may wish to explore further at least as an interim solution.

Implementing a Linux Solution

In order to assist others in putting together a Linux solution, I have put together a list of tips and pointers to give you a good starting point. Some of these areas are discussed further in the wonderful white paper by Caldera, Inc., “Using Linux in a Commercial Setting.” The primary focus of your effort is probably to convince management that a freely available OS is a viable solution. This is rarely an easy task by any stretch of the imagination. If you have strings in the company, plan on pulling them.

Getting a Game Plan Together

Before presenting your case to management, be sure to have a game plan in order. Don't jump up and shout “Let's run Linux.” at the first project meeting. Corporate ties with commercial solution providers often run deep, so be careful. Put together a detailed implementation plan, complete with a cost savings analysis and time schedule. There are a number of things you can do to help yourself in this regard.

Actively research the necessary areas. Provide solid numbers for commercial solutions. Be sure that you have accommodated all aspects of the project within your proposal. Make sure that all issues of connectivity and software facilitation have been addressed. Think of it as a legal battle—leave no loopholes in your argument.

Obtain and maintain high-level contacts in the industry. Meet with other folks who have successfully implemented a Linux solution. They may be able to provide additional insight into your argument. Planning on running the latest and greatest version of “product X” under Linux? Chances are, someone else has already driven the Linux wagon down that road—investigate.

Establish a good flow of incoming information. Actively participate in the various Linux newsgroups. They are a wonderful resource for obtaining contacts and production information. Subscribe to *Linux Journal*. Helpful articles and vendor information are in abundance with each issue.

Hardware integration—make sure that your proposed hardware will function once its all together on-line. If you can't do it yourself beforehand, try to find someone who has. The worst thing in the world is to win the battle with management and run into hardware issues which require additional purchases to patch a problem that you didn't foresee.

Presenting your Solution

Once a solid proposal has been constructed, present your case. Try to leave a solid impression of Linux with your attendees. Some key Linux points to hit on include:

- POSIX compliance
- 32-bit architecture
- Cost savings
- Availability of tools and software solutions
- Network capabilities (TCP/IP, IPX, AX.25, etc.)

Unfortunately, most folks still perceive Linux as a toy. In your presentation, be sure to point out the efforts of major industry players, such as Caldera and WordPerfect. Let them know that Linux is quite capable of providing a solid solution for your organization.

Conclusion

Linux is a viable solution platform. Our nationwide network of training centers is a living testament to that statement. With the proper direction, its proliferation in the workplace can continue on an upward trend. The Linux operating system reminds me of an expansion baseball team. It has a lot of fans, but nowhere near the fan base of an established and proven team. It is young and full of promise, and one day, just maybe, it will win the pennant.

Scott Burkett is a full time C/Unix technical instructor for Decision Consultants, Inc. (DCI) (http://www.dcicorp.com/), one of the country's largest software services consulting firms. He has worked with a variety of languages on multiple platforms. Scott is one of the co-authors of the of *The Linux Programmer's Guide* , part of the Linux Documentation Project and the author of *The Linux Bootkit*. An accomplished webmaster, he has set up web sites for the Southeast Region of DCI (http://www.computerppl.com/) and The Tampa Bay Linux GNU Technical Society (http://www.intnet.net/). Scott can be reached through the Internet as burkebs@intnet.net.

Archive Index Issue Table of Contents

Advanced search

Programming with the XForms Library

Thor Sigvaldason

Issue #41, September 1997

Part 3 shows us the means to give our game simulator a more professional appearance and to add a few goodies.

In the first two articles of this series, we learned how to install XForms and began building an application (a game theory simulator). In this final article, we spruce up our program and look at a few XForms features that we skipped last month. As always, source code and further information can be found on the home page for this series at http://a42.com/~thor/xforms.

Review of Our Progress So Far

If you've been following things reasonably closely, you probably remember the general outline for constructing an XForms-based application:

1. Include forms.h to access the XForms routines
2. Call **fl_initialize()** as soon as possible
3. Set up your graphical interface by creating forms
4. Assign actions to relevant objects by setting callbacks
5. Show one or more forms
6. Turn control over to **fl_do_forms()**

Last month, we followed this procedure to get our basic game theory simulator up and running. While that gave us the basic windows we needed to be able to control and observe the underlying simulation, there were a number of shortcomings. We had no way to save the settings of any particular run of a game, no pull-down menus and no pixmaps to make our program look somewhat professional. The new version of the simulator (called xgtsim2) adds all of these elements and includes a few other bells and whistles that we discuss throughout this article.

The basic approach, however, has not changed. If you could follow last month's source code, you should have no difficulty understanding xgtsim2, even though it is a larger piece of software. The core of the program is the same, since all we have done is add a few more features. Extra features make a program useful, and one of XForms's great strengths is in providing straightforward methods for enhancing a program's usability.

A Look at xgtsim2

You can find the source code for xgtsim2 in Listing 1 on the home page, but it will save you a lot of time if you download it from the web site. Saved as an ASCII file with the name xgtsim.c, it compiles with the command:

```
gcc -lX11 -lforms -lm xgtsim.c -o xgtsim
```

From within the X Window System, you should be able to run the program by entering **./xgtsim2** in whichever directory you compiled it in. The running program should look something like Figure 1.

Figure 1. Screen Shot of xgtsim2

As noted near the top of the source code, everything that has been added to xgtsim2 since last month's program (i.e., the original xgtsim) is marked by the string ***NEW***. This should make it a little easier to find the segments of code we discuss below.

New Items

The first thing to notice when xgtsim2 runs is that the starting window (**main_window**) is a little larger than it was last month, and it now includes three pull-down menus (**File**, **Settings** and **Help**). Adding these menus with XForms is quite straightforward. If you look at the code for the **create_forms()** function, you'll see we have added an **FL_UP_BOX** to hold the menu items, then call **fl_add_menu()** three times. The inclusion of the **FL_UP_BOX** isn't strictly required, but it does make our menu area stand out from the rest of the main window.

In the first of the **fl_add_menu()** calls, we create the file menu. Note that this involves only a single **FL_OBJECT**, with the menu entries (e.g., **About**, **Load**, etc.) being assigned to this object with the use of the **fl_addto_menu()** function. That is, we need to assign only a single callback for the entire menu, which we do by telling XForms we want the function **file_menu_routines()** to be called whenever any entry in our **File** menu is selected by the user.

A little earlier in the source code, we can see that **file_menu_routines()** uses a call to **fl_get_menu()** to figure out which of the four possible entries was actually selected. It then calls the appropriate function. If, for example, the user selects **About** from the **File** menu, XForms knows it must execute the code included in **file_menu_routines**, because that's the function we have assigned with the callback. In calling this function, it passes the file menu object to the routine, so the call to **fl_get_menu()** returns a 1 (since **About** is menu item 1).

We use a very similar structure for the **Settings** and **Help** menus, creating a function to handle the callback for each (**settings_menu_routines()** and **help_menu_routines()**). With just a few lines of code, we have added a fairly complete menu system to the program. XForms does most of the hard work here, such as actually drawing the menus when the user clicks on them, highlighting the entries and so on. This leaves us free to focus on the underlying flow of the program.

One common element of menu design we have not implemented is the use of keyboard shortcuts. XForms allows for these accelerators via the **fl_set_object_shortcut()** function, as well as offering mechanisms to grey out menu entries under certain conditions, change the visual look of the menus and so on. More information about these routines can be found in the XForms documentation (see Resources).

Other additions to **create_forms()** are calls to **fl_set_object_resize()** and **fl_set_object_gravity()**. The easiest way to see what these do is to run the original xgtsim and resize the main window with the mouse pointer. If you do this, you'll notice that the buttons always grow at the same rate as the overall window; make the window really big, and the buttons become enormous. It is not very attractive, so we want to use the **gravity** and **resize** parameters to improve this behavior.

Almost all graphical elements in XForms have a default **resize** setting that causes them to grow in direct proportion to the window in which they were created. We change this behaviour in **create_forms()** by calling **fl_set_object_resize()**. This function takes two parameters: the object to which it applies and a setting value, which can be **FL_RESIZE_NONE**, **FL_RESIZE_X**, **FL_RESIZE_Y** or **FL_RESIZE_BOTH**. For the **FL_UP_BOX** that holds the pull-down menus, we use the **FL_RESIZE_X** option, since we want the menu bar to always be the width of the screen but maintain a constant height. Similarly, we use **FL_RESIZE_NONE** for the buttons so that they remain the same size no matter how the window is changed.

Object gravity is a related concept and determines how objects should be oriented to the window in which they are drawn. Using the example of the

menu bar again, we don't want the menu to drift down at all, even if the user resizes the window to be very large. The function **fl_set_object_gravity()** requires a parameter for the relevant object and two subsequent values, which dictate orientation behaviour. The first of these determines which direction the upper-left-hand corner of the object should move as the underlying window is altered. The second sets the behaviour of the lower-right corner. Since we always want the **Help** menu to appear on the right edge of the menu bar (and stay at the top of the window), we use the following form of the function:

```
fl_set_object_gravity(obj, FL_NorthEast, FL_NorthEast)
```

Conversely, we want the **File** menu to stay to the left, so we replace both occurrences of **FL_NorthEast** with **FL_NorthWest** in that call. With a little thought on the programmer's part, XForms makes it quite easy to have windows that resize in attractive ways. This can add a significant amount of polish to any graphical application and make it usable in a wide variety of circumstances.

To dress up our program a little, we have inserted a pixmap in the **About** window. The actual pixmap data is stored in a variable called **xgtsim_logo** which is included at the end of the source code. We then need two calls to create the object that holds the pixmap and assign our data to it:

```
obj = fl_add_pixmap(FL_NORMAL_PIXMAP, 13, 13,
        70, 55,"")
fl_set_pixmap_data(obj, xgtsim_logo)
```

We declare how much space we need for the pixmap image in the **fl_add_pixmap()** call, but it is the second function which actually assigns the data. Since this pixmap is just for decoration, we don't need to declare any callbacks. To use pixmaps as buttons (in the same way that programs like Netscape do), you'll want to have a look at the **fl_add_pixmapbutton()** function in the XForms documentation.

The fact that the logo is not particularly artistic should not be taken as a shortcoming of XForms. I am reasonably competent at hacking out C code but, even when equipped with *The GIMP*, I'm no Picasso.

Using Goodies

With the inclusion of a menu system, resizing parameters and some decorative pixmaps, XForms-based applications like xgtsim2 can be easily polished into user-friendly, attractive software. XForms also provides a slew of easy-to-add program elements called “goodies”.

An example of a goodie occurs in the code for **help_menu_routines()**. If the user selects **Use** from the help menu, he gets a window that displays information about how to use xgtsim2. Since the program is just an example, we haven't actually written any help files for it—we just want to display a window explaining that help is not (yet) implemented. We could create this window manually, adding some text objects, an “OK” button and so on; however, this is a lot of work just to say there is no help available. Instead, we use a goodie called **fl_show_alert()**. This function accepts three lines of text as parameters, as well as an integer value to determine the placement of the ensuing announcement (the value 1 just tells XForms to place the window in the center of the display). With one line of code, we have an easy way to display text messages to the user without having to design a new window ourselves.

An even more powerful example of a goodie is the XForms-supplied file requester. Writing one of these from scratch can take a good deal of time, since we would need to create a window with some kind of browser, open and close buttons, implement a filtering mechanism, etc. The **fl_show_fselector** does all of this for us and allows the **load_config()** and **save_config** functions in xgtsim2 to be very compact. The full form of the function is as follows:

```
fl_show_fselector(const char *message,
        const char *directory,
        const char *pattern,
        const char *default)
```

The four string parameters allow us to set the selector's message, a specific directory to start from, a filtering pattern, and even a default file name. All of this occurs with a single function call. A somewhat subtle feature of the file selector is the existence of six such selectors, each of which remembers the last directory if the ***directory** string is passed as length 0. In xgtsim2, we use two of them, one for loading and one for saving. In each case, we declare which selector appears by making a call to **fl_use_selector()** before calling **fl_show_selector()**. That way, if users are loading data from one directory and saving it in another, they will not need to keep clicking back and forth between directories each time they want to access files.

There are also mechanisms for adding configurable buttons to the selector, setting the window title, and so on. Anyone who has designed a method for letting users load and save files will appreciate the amount of thought and planning that has gone into this widget.

There are many other goodies provided by XForms, including routines to get input from the user (**fl_show_input**), other message display routines (**fl_show_question**) and even a quick and easy method for getting color selections (**fl_show_colormap()**).

On Your Own

There is still much about XForms that we haven't touched on in this series, but the documentation included with XForms is excellent at explaining all of the resources available. With a little effort on the programmer's part, the library provides for fast program development and a professional look. There's even a form designer included in the XForms package which enables you to design an interface using a mouse. This makes creating complex windows a breeze, and the software produces output which can easily be incorporated into your source code.

Even if you never create a “killer app” with XForms, the basic lessons of placing GUI elements, assigning callbacks and showing windows are reasonably transportable to other programming environments and libraries. These articles should give you the basic knowledge required to create X programs. To paraphrase Donald Knuth, go forth and create great software.

Resources

Thor Sigvaldason is the author of the statistics program xldlas which uses the XForms library (see *Linux Journal*, February 1997). He is trying to finish a PhD in economics and can be reached at thor@netcom.ca.

Archive Index Issue Table of Contents

Advanced search

Packet Radio Under Linux

Jeff Tranter

Issue #41, September 1997

Linux and ham radio share a common spirit of cooperation, experimentation and do-it-yourself attitude. These two interests come together in packet radio.

In my teens I spent many enjoyable hours tinkering with radio equipment and communicating with other “hams” around the world through the medium of amateur radio. Moving away to attend a university and start a career and family meant that my hobby had to go on temporary hiatus for a number of years. Recently, I came back to the hobby and decided to explore an area of amateur radio that didn't exist in my teen years—digital packet radio. As an avid Linuxer, I was intrigued to see how I could use my Linux system for packet radio. I was on a limited budget, and wanted to get started without investing in a lot of hardware and software.

What Is Ham Radio?

Amateur radio is a pursuit enjoyed by millions of “hams” around the world. By international agreement, most countries have allocated a portion of the radio spectrum for amateurs to experiment with radio technology. The hobby goes back to the early days of radio and is popular with people of all ages. Operating an amateur radio station requires an operator's license, which can be obtained by passing an examination that covers radio theory, regulations, operating practices and basic electronics. Full privileges also require a knowledge of the International Morse Code (yes, it is still used) although some countries now offer no-code licenses that typically include restrictions in operating modes and frequencies.

Hams are known for building their own equipment and accessories, experimenting with new technologies and helping each other and the public. This is close to the spirit of Linux, so it is not surprising that many hams are also Linux users.

What is Packet Radio?

As personal computers became increasingly powerful and affordable, amateurs looked at using radio for digital communications. Packet radio is one such method in which text is encoded as binary data and transmitted via radio in groups of data, called *packets*. One popular protocol developed for this purpose is AX.25. Based on the X.25 protocol but adapted for the special needs of amateur radio, it offers error-free, packet-based communication between stations. Other protocols, such as TCP/IP, can run on top of AX.25. A few of the applications of packet radio include chatting by keyboard with other hams in real-time, using packet bulletin board systems, sending electronic mail, and connecting (via gateways or worm holes) to other packet networks or to the Internet. Data rates range from 300 bits per second on HF (High Frequency) bands to 1200 bps on VHF (Very High Frequency) and 9600 to 56Kbps and beyond on UHF (Ultra High) frequencies.

What Kind of Packet Radio Hardware Do You Need?

A typical packet radio station consists of a computer or terminal connected to a Terminal Node controller (TNC), and radio transceiver (transmitter/receiver). A TNC is a device containing a small microprocessor, dedicated firmware in ROM, and a modem to convert signals back and forth between audio and serial bit formats, and also encodes and decodes data with the AX.25 protocol. A typical TNC connects the radio to the computer via its serial port.

Another popular option is the “poor man's modem-only” Terminal Unit (TU). It essentially has a modem chip sandwiched between its audio and serial port connectors. Software drivers on the computer must therefore handle all of the AX.25 protocol. It's popular because of the price—typically one third the cost of a TNC.

The typical DOS-based packet setup has a number of limitations. Since the system is single user, the computer must be dedicated to packet and cannot easily be used for other purposes. Generally, the computer is used as a dumb terminal or dedicated software is used that takes over the whole computer. Although some software packages such as JNOS run on XT or AT class machines, Linux typically runs better on a 386 machine having limited memory than a commercial operating system alternative such as Windows 95.

Packet software is either free or distributed as shareware in binary form. Some packages, notably JNOS, are also available as source code.

What Does Linux Offer?

As of release 2.0, Linux has native support for the AX.25 protocol built into the kernel (a unique feature among operating systems). For earlier releases, Alan Cox's AX25 package is easily patched in. Furthermore, AX25 is integrated with the rest of the Linux networking code and utilities. To Linux, a packet radio interface appears as just another network interface, much like an Ethernet card or serial PPP link.

The kernel contains device drivers for serial port TNCs as well as several popular packet modem cards. It even offers a driver that uses a sound card as a packet modem (more on that later).

Once up and running, you can let users telnet into your Linux system via packet radio, offer them a Unix shell or one of several BBS programs, or even let them surf your system with a web browser. Thanks to Linux's multiuser and multitasking capability, this occurs without affecting the normal use of the system.

A Linux machine can act as a router to connect packet network traffic to a LAN or Internet connection and can route e-mail via packet. You can have multiple packet interfaces with many simultaneous connections over each interface.

Back to My Story

My first introduction to packet was using the sound card modem driver with a hand-held 2-meter band transceiver for 1200bps packet. The audio connects from the PC sound card to the radio's microphone and speaker jacks. A signal from a serial or parallel port in conjunction with a simple (one transistor) circuit is used to control the radio's PTT (Press To Talk) circuit. This approach requires no TNC and no packet modem—if you already have a computer and sound card this costs almost nothing. Figure 1 shows a block diagram of my setup.

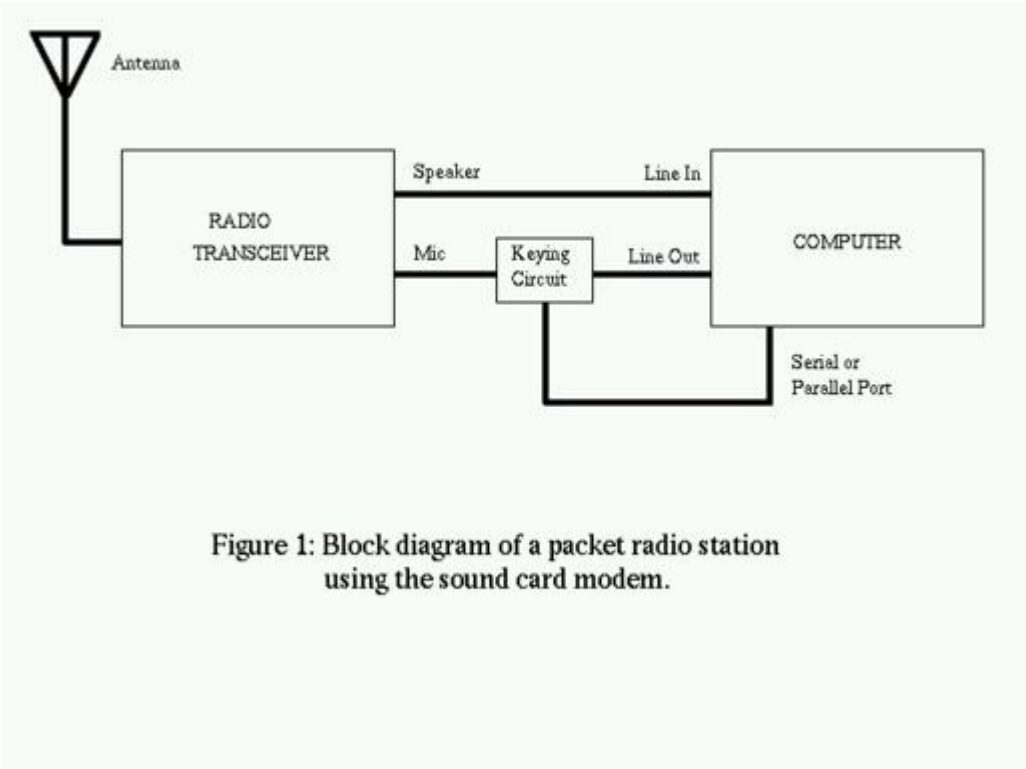


Figure 1: Block diagram of a packet radio station using the sound card modem.

Configuring the system was straightforward—I just followed the detailed instructions in the AX25 HOWTO. Utility programs included with the AX25 package allow me to monitor the packets being broadcast and received, and to call and answer other packet stations.

I obtained an IP address from the local IP coordinator (the 44.x.x.x ampr.org Internet domain is assigned for packet radio) and configured my system for TCP/IP over packet. All the standard network tools then operated over packet. Assuming they are configured for TCP/IP, I can **ping** or **finger** other stations and connect to them using **telnet**. Similarly, I can log on to my home Linux machine over the Internet.

Next I plan to set up a simple BBS system users can log on to via packet radio. I'd also like to look at more sophisticated packet networking tools supported under Linux, such as NetRom, NOS and Rose. In the future I may even explore options for higher-speed packet such as the 56 Kbps Ottawa PI2 card.

Conclusions

As well as being fun, packet radio under Linux taught me a lot about networking, much of which is also applicable to Ethernet, X.25 and other network protocols.

Linux is a great platform for packet, particularly since it is fully integrated with the rest of the networking subsystem. Its reliability lets you leave a Linux system up for long periods of time without crashing (ideal for a BBS environment). As an example, one local Linux system has been on the air

continuously for over 310 days without interruption. Some of the software I used was still in alpha release yet was stable enough to use. Finally, packet radio has opened up a whole new area of Linux for me to explore, ensuring that I won't run out of things to do in the foreseeable future.

Acknowledgments

Thanks go to Gord Dey, VE3PPE, for reviewing this article and adding many valuable suggestions. I also wish to thank Terry Dawson for writing the AX25 HOWTO and utilities, Alan Cox, Jonathan Naylor and others for writing the Linux packet code and Thomas Sailer for writing the sound card packet modem driver.

Resources

Jeff Tranter is the author of the Linux Sound and CD-ROM HOWTOs and the book Linux Multimedia Guide, published by O'Reilly and Associates. His hobbies beyond Linux and ham radio include playing guitar, cross-country skiing and lava lamps. His ham radio call sign is VE3ICH and you can reach him via e-mail at jeff-tranter@pobox.com.

Archive Index Issue Table of Contents

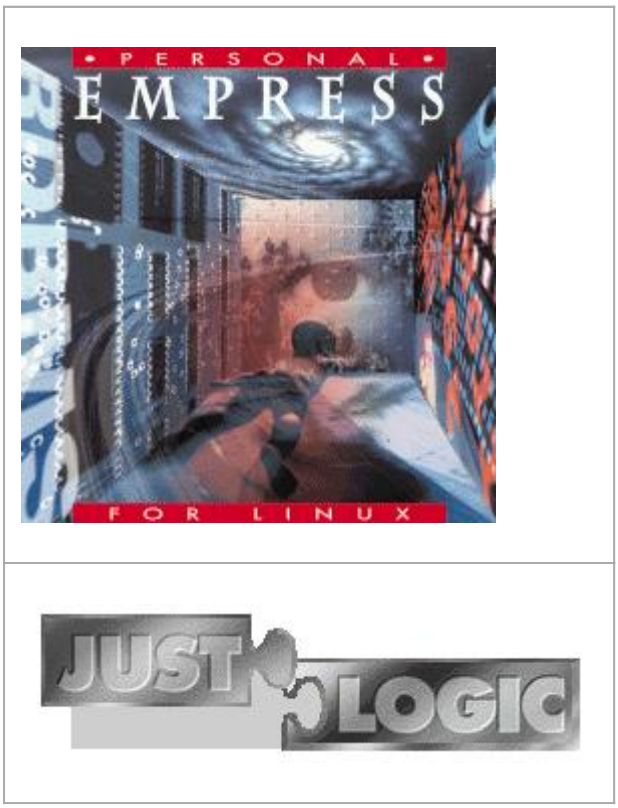
Advanced search

Product Reviews: Empress RDBMS and Just Logic/SQL RDBMS

Rob Wehrli

Issue #41, September 1997

The bottom line in choosing between these two Linux database packages is one of cost versus time and user programming capability.



- Manufacturer: Empress Software Inc.
- Phone: 301-220-1919
- E-mail: sales@empress.com
- URL: http://www.empress.com/

- Price: $1500.00US

- Manufacturer: Just Logic Technologies Inc.
- Phone/Fax: 800-267-6887; 514-642-6480
- E-mail: info@justlogic.com
- URL: http://www.justlogic.com/
- Price: $295.00 US
- Reviewer: Rob Wehrli

In my quest for a formidable relational database management system that would run under Linux I came across a pair of applications that fit my wish list very well. I needed a database that was fast and easy to use, set up and manage. The system had to work in an Internet/Intranet environment, support multiple users and be vastly configurable. The cost had to be within my budget.

I tested Empress RDBMS first. Marketed for Linux, it includes several features that make it a clear choice for discriminating individuals and businesses who can afford it. I then tested Just Logic Technologies' Just logic/SQL RDBMS, a product which does not include as many bells and whistles, but does offer core functionality for about a third of the price of Empress.

The first thing I noticed when the Empress package arrived at my door was 18 bound and 7 unbound (loose-leaf) manuals. This documentation set is fully cross-referenced and includes titles for the core server installation, server management and administration, client utilities, 4GL application development platform, web server interface, ODBC driver, SQL reference, GUI Builder and much more. The manuals are printed in easy-to-read fonts with page numbers in bold and chapter data included on every right-side page. Each manual includes a complete index and a diagram with the entire documentation roadmap directing attention to the order in which manuals should be consulted. Empress gets a resounding “A+” for their documentation and additional kudos for complete man pages that complement their hard-copy documents.

Installing Empress from the installation diskettes requires basic Unix system administration experience. While the installation documentation is complete, the diskettes included a broken **cpio** command on the label. Nothing too difficult to overcome—merely an annoyance in an otherwise outstanding presentation. Installation of the Empress GUI Development and Runtime requires installation of Motif (libXm) libraries and knowledge of the path to their locations. Unfortunately, I was unable to test this feature of the Empress product bundle as my a.out-based Motif libraries were not recognized by the installation program.

Dismissing the minor installation difficulties and getting to the meat and potatoes of testing Empress left me pleased with the package contents. Several utilities, such as the interactive SQL interface and the dBase file import and export programs, provided me with considerable appreciation for the talent and foresight of the Empress development group. While every SQL interface is "interactive", Empress is truly interactive in that it is capable of, among other things, prompting the user during table design for specific table attributes and for related variables. This is enough to excite even the most placid DBA. If you use the **empsql** interface for inserting data, it prompts for user input and is useful as a front end for data input by non-programmers.

Plugging Empress into my particular application required little more than prototyping a database in Access 7.0 and using the ODBC driver to export the tables to Empress. This worked with numeric data types quickly and efficiently. Unfortunately, the Access text data type produces problems when exporting tables from Access to an Empress database. I did receive a prompt reply from Empress' e-mail support claiming that this anomaly is due to different definitions of text data types between Access and Empress and that Access cannot export text attributes to Empress because of this dissimilarity. It seems to me that Empress should develop a workaround for developers using Empress ODBC and Access for database prototyping. Working without a text data type is not a viable option. One solution for those DBAs using Access is to export data from Access to a dBase or comma-delimited text file, then import to Empress using their fine import utilities. I also found no support for **varchar** data types.

The first databases I built were simple tests to see how well the Empress utilities produced desired output. Empress performed flawlessly, and the Interactive SQL tool is a real glowing ember in a crowded fireplace of functional components. Their **empsql** and supporting configuration files allows for custom user configuration, much the same as configuring an e-mail reader. For example, I selected **joe** as my SQL editor instead of the default **vi** for console-based edits.

Testing the speed of the database with data imported from a combination of Empress utilities was very straightforward. I decided that several joined tables and multiple nested queries would provide a good performance test. Empress produced results far above my expectations. I was suitably impressed with the raw speed at which Empress rushed data back to the screen. A search of 810,000 records, where several calculations, conversion of data types and summing and ordering of resultant sets was required, completed in less than 15 minutes. By comparison, the same query on an SQL server machine(1) took about 28 minutes.(2)

Incorporation of an Empress database into a web environment is accomplished without hassle using their DataWEB package. Writing HTML forms with Empress extensions to query the database is straightforward for anyone with a little HTML experience. My Red Hat 4.0 system, installed with the supplied Apache httpd server, integrated quite nicely.

My final test, which says as much about Linux as it does about Empress, included flipping the power switch to the off position in the middle of a query. After several minutes of waiting while fsck fixed my purposely distorted file system, Empress recovered without any noted glitches. Of course, I was hoping to crash the database to test the on-line backup utility, but it would not crash. I didn't get mad—I got even. I deleted the database, and it restored quickly and without incident. I was ready to query once more.

Empress also includes a report writer that I was unable to test due to scheduling limitations.

Lacking in Empress is much of what is lacking from many commercial RDBMS products today, full SQL-92 support. A point-and-click management tool would be nice and even recommended, since it is standard fare with Microsoft's SQL Server. Considering the cost of the Empress package that I tested is about $1500, it is a bit expensive when compared to the cost of a typical Linux distribution. However, the ease of use of its utilities and the completeness of its documentation man pages and on-line help make it a good choice in the professional world. It is a remarkable product that will benefit the many Linux users who find it a perfect fit for their needs.

Just Logic

The next candidate in this database duo is Just Logic/SQL. It is compact, easy to install and use and extremely cost-effective. Priced at $295 for an unlimited number of users, I found it to be in a value class of its own. Many of the utilities and niceties included in the Empress package are not found in the Just Logic/ SQL product, but it features a simple and effective SQL interface and robust C, C++ and pre-compiler interfaces. The sqlweb interface for putting databases on the Net is an option value-priced at $175.

Installing Just Logic/SQL was as simple as can be. Perhaps the most complex component for newcomers would be creating a user account and group for the server/administrator, which is thankfully a point-and- click operation in the Red Hat Control Panel. Since I downloaded the trial version of Just Logic/SQL from their web site, the documentation was in Adobe PDF files, which require an appropriate reader before beginning installation. I liked the searchable PDF files.

JTL comes with a test database and a warning that it may take a few minutes to install depending on your hardware. The sample databases are included in three different formats, each serving as learning examples of how to use the C, C++ and pre-compiler interfaces. I chose the C version, which installed a small database in less than two minutes. I assume the warning must be a holdover for 386 Linux users. I tried the same file on a 486-66 with 32MB and a Seagate fast SCSI-2 hard drive on a 16-bit Future Domain controller, and it took approximately three minutes, certainly not as long or as involved as compiling a kernel. You can probably safely ignore the warning if you are currently driving any hardware combination built after the Reagan administration.

The documentation provides a simplistic schema for the “abc” sample database. Something everyone can appreciate is database guru Joe Celkos' naming conventions—table names are plural and in uppercase, attributes are singular and lowercase. The Just Logic/SQL sample mixes conventions just enough to be annoying, but this is rather common in an area where MS-Access-based converts excel (pun intended).

During testing this product performed very well. Using the same 810,000 record database and query, it brought back answers in just under 13 minutes. However, the slight difference in performance is less significant when compared with the time it took to get the data into the package. While Empress import utilities handled things in just a few minutes, I spent about 45 minutes writing a C program to import the data into Just Logic/SQL. Just Logic/SQL supports **varchar** data types.

Using Just Logic/SQL in a web environment was another exercise in simplicity. Installation and setup are a matter of copying the executable into your cgi-bin directory and editing a sample configuration file with your system details. The executable must set UID to the database owner, which is accomplished easily with the documented commands. The **sqlweb** documentation is clear and concise. Creating HTML pages for database access using **sqlweb** is well-defined with complete examples in the sqlweb.pdf. I was able to access data from the Linux/Just Logic/SQL/Apache combination from my Linux Netscape browser within minutes. It is quite exciting to see how fast it works. I spent several hours writing complex queries and HTML forms pages to see if I could break it.

Just Logic/SQL is perfect for low-budget shops who need a relational database solution. Students and professionals will appreciate it for its simplicity and robustness. The SQL, web, C, C++ and pre-compiler interfaces offer enough choices that anyone can immediately begin using the product to store and manage data resources. I heartily recommend it for anyone with some programming experience. The many examples of coding provided on the Just Logic/SQL web site is a fine starting point.

Conclusions

The bottom line in choosing between these two Linux database packages is one of cost versus time and user programming capability. Both packages offer programmers the flexibility to do just about anything they wish with their respective systems. Both provide data control, manipulation and administration. Both are performance-oriented and presented no problems as delivered. Both worked well in networked environments; however, I was unable to test either of them in a busy multiuser setting. The only thing I would want from either package is conformance to the SQL-92 standard, although both currently claim SQL-89 conformance. Both vendors have substantial quantities of information available on their web sites. I found both of these packages surprisingly supple and responsive, easy to install, configure and run on Linux with basic Unix skills. I think you will, too.

Test Platform

Resources

Rob is a systems engineer and longtime resident of Honolulu, Hawaii. He enjoys playing golf and chess. He can be reached at rowehrlii@pixi.com.

Archive Index Issue Table of Contents

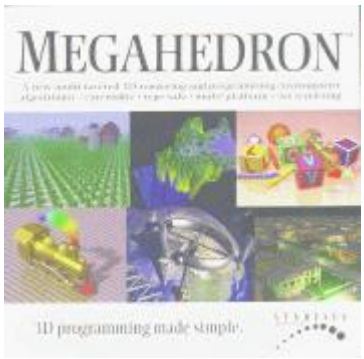
Advanced search

Megahedron—A 3D Graphics Environment

Michael J. Hammel

Issue #41, September 1997

Megahedron is a modeler and 3D graphics engine that uses an interpreted language similar to POV-Ray's scene description language.



- Manufacturer: Syndesis Corporation
- E-mail: syndesis@threedee.com
- URL: http://www.threedee.com/
- Platforms: Intel Linux, Silicon Graphics, Windows NT
- Price: $99US
- Reviewer: Michael J. Hammel

The world of 3D graphics on Linux has come a long way in the past 2 years. When I first started investigating graphics tools for Linux, there were only a handful of 3D renderers publicly available and almost no 3D modelers. Since then the number of modelers has grown significantly (I can count 5 full-blown modelers currently working plus at least 2 others in development). 3D rendering tools have also seen a vast increase, with POV-Ray and BMRT (Blue Moon Rendering Tools) two of the best modelers available for any platform, heading the list.

Not long ago I came across a new product I had seen announced in the comp.os.linux.announce newsgroup: Megahedron. I hadn't actually used the

product (it was, and still is, a commercial product) but I was intrigued by the announcement. I had Megahedron on my “list of things to purchase”, when *Linux Journal* beat me to it and provided me with a copy to review.

Megahedron is a modeler and 3D graphics engine that uses an interpreted language similar to POV-Ray's scene description language. It differs from POV-Ray in a numbers of areas, such as the ability to do wireframe animations on the fly and built-in network rendering. The package is supported on a number of platforms, including Windows NT (x86 and DEC Alpha), Silicon Graphics and Intel Linux. The $99 list price gets you a CD-ROM containing binaries and complete configurations for each of these platforms. Licensing covers any machine the purchaser uses with Megahedron, but if another user wishes to use it, he must purchase his own copy. It's a fairly unrestrictive license, as far as commercial products go.

Installation

The distributable package consists of a single CD in the customary plastic casing. All documentation is in HTML format on the CD. There is a single insert on the CD explaining where to begin in the documentation contained in an HTML file in the root directory named mhedron.htm. This page is a master Table of Contents for the complete documentation.

Installation of Megahedron is simple:

1. Choose a base directory in which to install the package, generally, /usr/local. If the directory does not exist, create it with **mkdir**.
2. Copy the tar file from the Linux directory on the CD to the chosen base directory.
3. Unpack the tar archive.

The only problem with installation is that the instructions are listed in the fourth part of the fourth section of the first chapter of the Manual. The Manual is the third heading in the Table of Contents. The second entry in mhedron.htm is a “Quick Tour”. This tour suggests making changes to source code, which can't be done directly from the CD, so you must first do the installation. The installation also says to “decompress [the package] from your hard drive or directly from the CD.” This is not quite correct since the package is not compressed—it's simply a tar archive. Despite these oversights, the installation is relatively straightforward.

Documentation

Unlike many of the free packages available, Megahedron comes with oodles of documentation, all of it on-line on the CD and formatted in HTML. This is a nice

bonus, since you can print the pages of interest from your browser. Most of the HTML documents print out as no more than 7 to 10 pages, which isn't too bad. Most impressive is the sheer amount of information provided.

The master Table of Contents contains links to five other areas: an introduction, a Quick Tour, an art gallery, the user's manual and sample source code. The Quick Tour provides a glance at some of the modeling, animation and procedural aspects of the interpreted language, called SMPL (Simulation and Modeling Programming Language). The tour is rather interesting. I found the train wireframe-animation particularly interesting, since most other tools I've seen don't offer such features (a notable exception is the **rendribv** program in the BMRT distribution).

The art gallery is not very impressive from an artistic point of view. The images present the modeling capabilities of Megahedron much better than the shading capabilities; future versions should explore the various shading capabilities provided by SMPL. This lack may be simply because the people who created the images are more technically than artistically oriented, but really useful 3D images should present a good blend of technical and artistic aspects. Some of the images from the art gallery are shown in Figures 1 and 2. Listings for the HTML code that goes with them are not printed in this article but are available by anonymous ftp in the file ftp.linuxjournal.com/pub/lj/listings/issue41/2282.tgz

Figure 1. Megahedron Cactus

Figure 2. Megahedron Roadster

An attempt was made to color-code the sample SMPL source, but the result is somewhat limited. There is quite a bit of code for experimentation purposes, although I have to admit I didn't run much of it.

The meat of the documentation is in the manual, a seven chapter document plus an Index, Glossary of Terms and SMPL Grammar Appendices. The seven chapters cover modeling and rendering aspects in fair detail. I'd like to see these two areas broken out into separate areas; Megahedron has merged the modeler with the renderer just a bit too much for my taste. There is also a good deal of material covering the use of shaders.

Features

Megahedron is feature-rich. The section of the documentation titled “What is Megahedron?” gives a detailed list of features, including:

- The SMPL interpreted, procedural language
- A programmable shading language (which uses SMPL)
- Rendering modes from wireframe to full ray tracing
- Various projections, including a fisheye projection
- Simulation capabilities such as collision detection and ray intersection
- Network rendering

This is not the complete list, of course, but it should give you an idea of the range of capabilities available.

SMPL

SMPL is the programming language for Megahedron, and it looks a little like Basic. It's fairly intuitive if you are familiar with tools like POV-Ray or BMRT and with vectors, object primitives and transformations. If you are not, there are plenty of simple examples to get you started and the glossary and indices in the manual will help you find the way through the source code.

An SMPL program has the following general format:

```
do task2, task3;
<include files>
integer a;
procedure task1 is
        <declarations>
        <statements>
end;
procedure task2 is
        <declarations>
        <statements>
end;
procedure task3 is
        <declarations>
        <statements>
        task1;
end;
```

Declarations include data, type and subprogram declarations. Statements include object declarations and instances, transformations and so forth.

The data types supported by the language are fairly intuitive, but are slightly different in syntax from RenderMan and POV. In the RenderMan RIB file format, a string type is a **String**, whereas Megahedron uses **char**. As you can see, the data type is obvious. RenderMan uses **point** to specify a set of 3 points in space, each of which is a float value. Megahedron uses the type **vector** for the same thing. POV, on the other hand, doesn't really have data types. All variables have declared values that get preprocessed before the code is processed, substituting the declared value for the name of the variable. Which method is better depends on the user's preference. I like defined data types because type checking can be done up front. Megahedron's use of the **const** statement allows for enumerated values, a nice addition that doesn't appear possible in a RIB file. (Although if you use the RenderMan API, you're writing C code, so enumerated types are not a problem.)

Subroutines are supported with the use of the **procedure** statement. Scoping of variables is much like C scoping, with variables accessible locally or globally, but not across procedures. Procedures are delimited with the **procedure** and **end**

statements. There is support for static variables in procedures as well as multidimensional arrays.

The language has one annoying aspect: it uses curly braces for comments. I've used quite a number of languages over the past 10 years and can't remember any that used curly braces for comments. The traditional C and C++ comment markers of **#**, **/**/** and **//** would, in my opinion, have been better.

One weakness in the documentation is the description of the file I/O routines. Although file I/O is possible, it's not clear how to output model information to a file. A few examples for outputting model information would have been a nice addition. After all, since I prefer using BMRT's renderer over Megahedron's, I need a way to output RIB files in order to use SMPL for modeling. Rendered images can be saved in RAW or TGA (Targa) formats. I found this information in the section on “Display Controls”, not in the section on “File I/O” as I expected. While perusing one of the system files, smpl_prims, I found that support is implied for the JPEG format as well, but I didn't find confirmation of this elsewhere.

Modeling Features

As with RenderMan's RIB and with POV-Ray, an SMPL file is a collection of sections describing a 3D scene. In SMPL you have sections for defining the camera and rendering options (similar to the sections outside the WorldBegin/ WorldEnd statements in RenderMan), object declarations, still frames and animation. Still frames are really the guts of the scene, where objects are instanced, textured, transformed and so forth. The proceduralism of SMPL allows for declaring objects once and instancing them many times throughout a frame. For example, a single ball might be defined as a sphere with holes cut out of it that can be used to instance 100 spheres in various states of unrest for a single frame.

An important part of any 3D rendering system is its ability to do transformations. Transformations allow an object (a sphere, box or more complex figure) to be moved to its location in 3D space prior to the actual rendering of the image, or to be sized or modified in shape (stretched or skewed, for example). Megahedron allows objects to be transformed relative to their current size and position or absolutely. Absolute transformations specify the exact size or position without regard to the current size or position.

Transformations are specified using the **with** clause for objects, as follows:

```
<object name>
                with
                        <transformations>
        end;
```

Transformations can be nested, and the relative transformation is based on a transformation stack, much like RenderMan or POV-Ray. Instancing an object gives it the current transformation state, and new transformations are made within the instance that apply only to that particular instance or any instances created below it. In other words, it's a hierarchical model. If you're familiar with POV or BMRT, you should have no problems learning the syntax and use of transformations in Megahedron.

One difference between RenderMan and Megahedron is how camera transforms are done. In the latter, the camera is actually moved. This is similar to the way POV handles camera transforms, but different from the way RenderMan handles it. It's important to understand what is being moved when using transformations in any 3D package.

Lighting primitives supported include distant point lights, spot lights and ambient lighting. These are the same lighting types supported by POV and RenderMan. Each type has its own parameters, such as brightness (known as “intensity” in RenderMan) and color. Lighting in RenderMan is handled through the use of shaders, so it's possible to create all manner of lights for use with BMRT. Megahedron appears to offer similar functionality, although I didn't delve into this area much.

One of the nicest features is the live animation capability. Wireframe displays, which can include hidden surface removal, can be run interactively. Interactive displays can also make use of mouse and keyboard input to control the display. There are examples provided that show exactly how this is done.

Programmable Shading

As with any 3D environment, modeling is not enough. Wireframe displays provide a glimpse of what your scene (or animation) will look like, but without extensive shading the model appears rather uninteresting. Megahedron provides a rich set of shading features: ray tracing using reflections, refractions and shadows, and image, bump and procedural mappings to name just a few. A collection of stock shaders such as granite, ridged and cloudy is provided. Targa (TGA) formatted image maps are supported. It's interesting to note that Megahedron maps all image map coordinates from (0,0) to (1,1), with the former being the lower left corner of the map. RenderMan maps (0,0) at the upper left and (1,1) to the lower right. If you're used to using BMRT, it is important to keep this in mind when using Megahedron image maps.

The shading language is closer to the RenderMan shading language than POV-Ray's texturing commands. This can be seen in the examples used in the “Anatomy of a Shader” section of the manual. Here colors are defined through the addition of values computed earlier, such as:

```
diffuse = (illumination + ambient) * color * .3;
specular = reflect (color * .4);
highlight1 = highlight1 * color;
highlight2 = highlight2 * color;
metallic_color = diffuse + specular + highlight1
        + highlight2;
```

This method of layering textures is different from POV-Ray's. In POV, an object's final texture is based on a series of textures defined within multiple texture{} statements. In a sense, the two methods are the same computationally, but from a users perspective the blending of layers is more apparent and under greater control in the procedural languages (Megahedron and RenderMan). If you started with POV, as I did, you may find this a bit confusing at first. Once you've learned how layering textures works with Megahedron, however, you'll appreciate the control it provides.

There are a number of predefined shaders provided in the distribution. In the system directory there are some extensions to SMPL (written in SMPL), including some interesting shaders such as “vampires”, which don't show up in mirrors, and “ghosts”, which only show up in mirrors. These shaders look rather interesting, and I have to wonder why the sample images don't appear to make use of them.

One important difference between the BMRT, POV and Megahedron feature sets is that only BMRT supports displacement maps. These are like bump maps except the point on the surface is actually moved instead of just adjusting the normals of the point to make the surface appear bumpy.

The Quick Tour

I like this aspect of the documentation, but it has some usability problems. First, the tour points to sample directories as **smpl/<directory>**. This is incorrect—there is no **smpl** directory. The tour also (unless I missed it) fails to mention that the system specific directories are the top level directories on the CD. Under the system directories is the mhd directory where the examples, code and system files are located. It can be a little confusing to find your way if all you do is read through the Quick Tour—be sure to look through the CD directory structure first.

Another rather interesting omission is that the Quick Tour doesn't mention the name of the program you will be running. After a quick search under the Linux directory on the CD, I found the bin directory and the i**mhd** program.

There were lots of problems running the examples. **ideal_gas**, **sonic_boom** and **slicer.smpl** are all examples that take input from the mouse. At first nothing I did seemed to affect the example; eventually, I found I had to move the mouse slightly for the button presses to be recognized. This might be an X-server

issue, but I think the problem is really the hot spot area defined by the SMPL code. It appears that the area that recognizes the button press is smaller than the visible button area.

While writing this review, I often switched desktops (I use FVWM2) between the xterm running my editor and another running Netscape in order to read Megahedron's documentation with the browser. I also ran the sample programs on the same desktop as Netscape. Whenever I switched to my editor desktop and then back to the Netscape desktop, some of the windows for the sample programs didn't get updated. For example, sonic_boom.smpl creates two windows—one that shows an aircraft and “sound waves” moving past it and another that shows controls for setting the thrust of the aircraft along with current speed displays. This second window would not be redrawn when I switched desktops. The ideal_gas example had similar problems, but I found I could get the control window to update if I forced a change to the display by moving the piston up or down. This limits the usefulness of the interactive programs with window managers that support multiple desktops.

Another minor nit is that windows created by the examples are not grouped. As a result, my window manager's AutoRaise feature does not raise all the windows associated with the example. This can be a little annoying, but it's a minor point.

Summary

Overall I found the feature set of Megahedron to be quite extensive. The documentation blows away anything else I've seen for a tool of this nature. I'm constantly looking for tools which can easily be picked up by a novice user. Megahedron certainly falls into this category simply on the strength of the documentation and sample programs. It is, however, not perfect. See Sidebars 1 and 2 for lists of Megahedron' pros and cons.

Megahedron Pros

Megahedron Cons

At one point the “Introduction to 3D Coordinates” says it is not necessary to know algebra or trigonometry to use Megahedron. Possibly true, but without either what you can do with Megahedron will be severely limited. Face it, knowing how to place objects in 3D requires not just an understanding of geometry, but trigonometry as well.

One area I didn't cover in this review is the rendering engine. I didn' do many full renders due to time constraints on my system. (I had other renderings running and needed to keep a little system time for other work.) If you get a

chance to try Megahedron and can compare it with BMRT and/or POV-Ray (or any other renderers that run on Linux) feel free to write it up and pass it on to me. Chances are good I'll include it in a future “Graphics Muse” column in *Linux Gazette*.

Despite the problems mentioned, I think Megahedron would be a good way for new users to get started with 3D images. Experienced users might find the animation capabilities quite useful as well, although the animations might not be as impressive with the builtin renderer. It's difficult to say without more detailed images that make full use of the shading language. The documentation is quite extensive and well written and the licensing is user friendly. I would recommend this package to anyone interested in learning more about 3D graphics.

Michael J. Hammel is an X Windows and applications software engineer for EMASS in Denver, CO. He is the author of the “Graphics Muse” column in the Linux Gazette, keeper of the Linux Graphics Mini-HOWTO and co-author of *The Unix Web Server Book* from Ventana. His interests outside of computers include 5K/10K races, Thai food and gardening. He suggests if you have any serious interest in finding out more about him, you visit his home pages at http://www.csn.net/~mjhammel. He can be reached via e-mail at mjhammel@csn.net.

Archive Index Issue Table of Contents

Advanced search

Solid Desktop 2.2 for Linux

Bradley J. Willson

Issue #41, September 1997

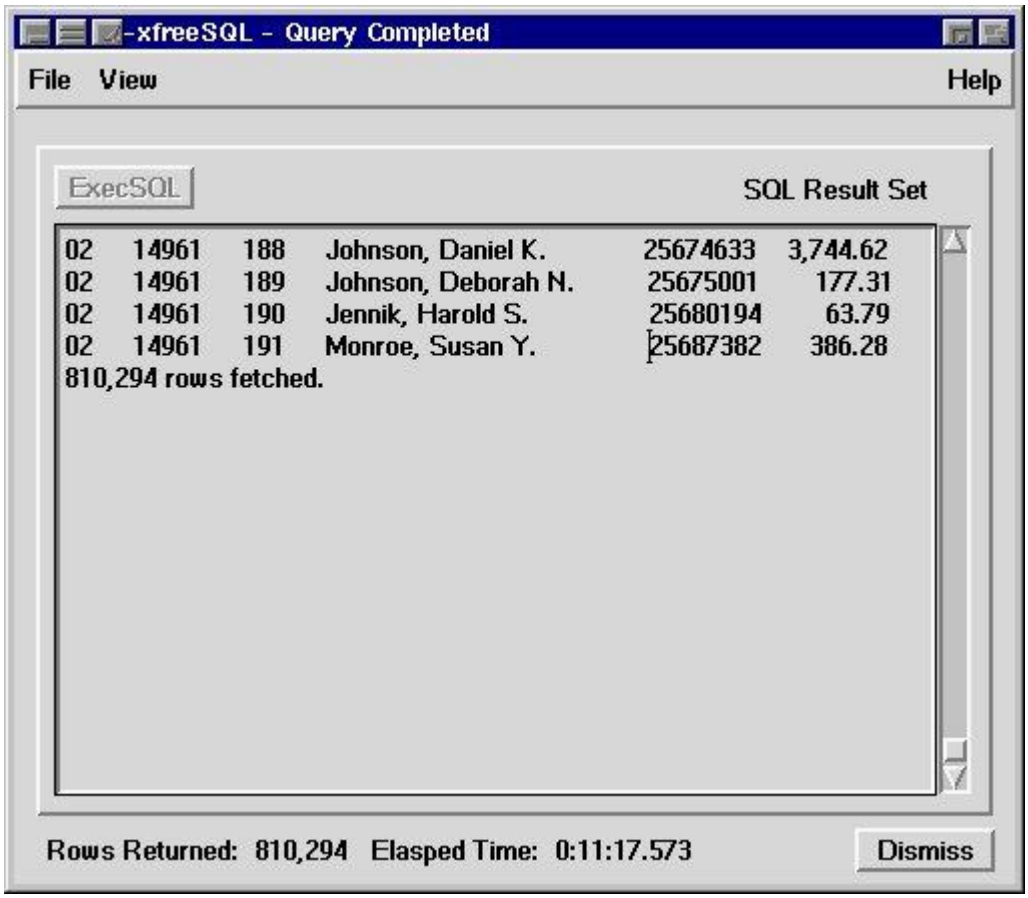
In the process of reviewing Solid Server, my understanding of database technology changed as my level of knowledge elevated.

- Manufacturer: Solid Information Technology Ltd
- E-mail; URL: info@solidtech.com; http://www.solidtech.com/
- Price: $99 US for single user
- Reviewer: Bradley Willson

When I installed Solid Server on my computer, I expected to find an application similar to Microsoft Access. Instead I found more “and” less. Solid Server has more raw “horsepower”. It is a race-ready engine waiting for a chassis. I discovered that Solid Server ships without the chassis. The task of “chassis building” (application development) is left up to the programmer. Solid Server's adaptability makes it easy to integrate into the customer's “chassis”. The end result is a database application that is capable of taking on the competition and winning.

In the process of reviewing Solid Server, my understanding of database technology changed as my level of knowledge elevated. I quickly realized I was dealing with a product possessing capabilities beyond my experience. Thanks to SSC's info@linuxjournal.com, I was able to recruit Rob Wehrli to help validate SQL standards and some of the technical aspects of the review. This review is the product of our e-mail collaboration across the Pacific. The screen-shots are from an interface application Rob is using with Solid Server.

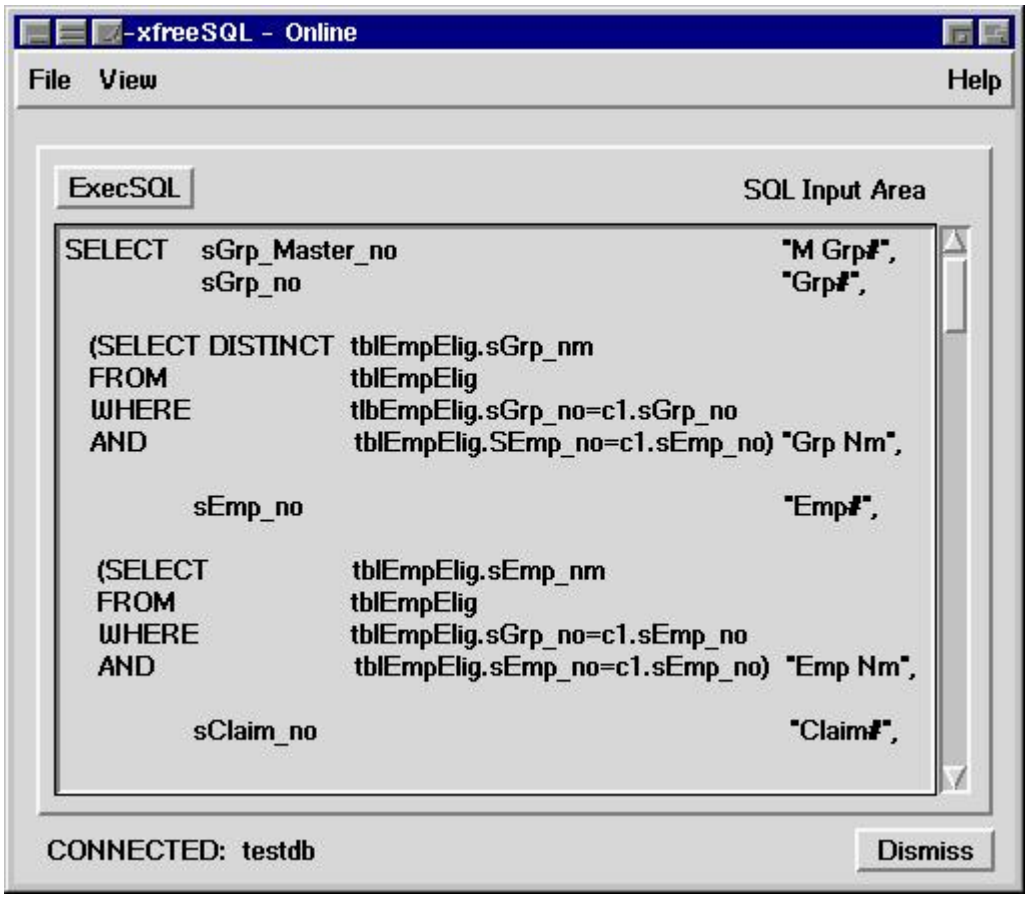
Solid Server for Linux 2.2 is powerful, extensible and extremely capable of serving database requirements ranging from a simple desktop application to a corporation-wide business integration. The combination of Solid Server's file handling capabilities (which reportedly extend into the 30TB range), multiple platform availability and extensive communication protocol suite makes Solid Server 2.2 an intelligent choice for a wide range of applications. I was unable to test the 30TB claim and networking capabilities because of my limited resources; however, judging from the list of companies using Solid Server, I am inclined to agree with these claims.



There is strength in numbers. Solid has ported its server to Linux and at least seven other popular platforms. Their installed base reads like a “Who's Who” in the global marketplace, giving one the feeling that this product is the proverbial “better mouse trap”. Names like Nokia, CompuWare and Tallahassee, Florida's Department of Highway Safety and Department of Motor Vehicles appear on their roster. Deutsche Telekomm, a major German telecommunications company, has purchased a large number of Solid licenses specifically for Linux. Solid Server is even embedded in Kone Elevator's control systems. Many of Solid's business customers and strategic partners have WWW sites, making it easy to gather references and additional information about the product. Kone Elevators maintains a web site at http://www.kone.com/ containing more information about their products and EleVision, their integration of Solid Server.

Communication is the heart and soul of Solid Server. For example, a database created on an NT platform can be moved, without conversion, to a Solid Server running on a Linux platform. Furthermore, the same database can be accessed by one or more client applications simultaneously, via any of seven common protocols, over a network or on one machine. The login screen-shot displays a simple interface to a networked database. Solid reports compliance with Entry Level SQL2 syntax, significant features from Intermediate Level SQL2, SAG CLI standards. The API also features 16 and 32-bit ODBC drivers and Solid's own SA interface, giving the customer the added flexibility to integrate Solid Server within existing structures with minimal conversion. The query screen shot

shows the SQL instructions used to extract the data shown in the result set example.



Data integrity is only as good as the steps taken to preserve it. Solid Server accomplishes this task using automated backup, concurrent logging and check points. Few applications can recover from a surprise shutdown, but Solid Server lost nothing during my testing. It is safe to say a stand-alone installation of Solid Server will survive most, if not all, system surprises. My system logs reported all of the recoveries as successful. Furthermore, the recovery process was so quick I hardly had time to read the extra line displayed.

This game belongs to the quick—Solid Server is fast. On a Pentium 75 with 20MB of RAM, it processed my small examples instantaneously. There was no appreciable degradation in performance as I added more and more information to the database. My test bed is humble by today's technology standards, but Solid Server performed beyond my expectations. It consistently returned result sets in a matter of seconds. Rob's result set screen-shot depicts results from a significantly larger database.

Assessing the quality of the documentation was an important aspect of my review process. Because my idea of a database was defined by Microsoft Access, I decided it would be wise to read Solid's documentation. The effort paid off and I gained a better understanding of how SQL and true relational databases work. The books and HTML documents are written with a focus on

Solid's Windows versions. As I read the documentation, I felt I was in familiar Windows territory, yet I was constantly translating terminology from the Windows environment to Linux equivalents. This is one aspect of Solid Server I would like to see improved. I recommend moving the referenced README files to the printed and HTML pages, merging them into the context as appropriate.

If you have questions, there are several options to explore. You can contact Solid's technical support through your local distributor who may offer a toll-free number or e-mail address. You can call Solid Information Technologies, Ltd. in Helsinki, Finland, or send an e-mail to their headquarters tech support. Solid does not offer a toll-free number. I sent e-mail directly to Solid on two occasions and received answers within 24 hours each time.

Solid Server 2.2 is value priced. The extensive list of features and capabilities give the Solid Server consumer an excellent return on his investment. Priced at three levels—$99 US per single user, $199 US per seat for multiuser and $495 US per server with Web- enabling option—this RDBMS is affordable for small businesses and large corporations alike.

Solid's decision to port to Linux complements the growing number of professional mainstream products that make Linux a respected contender for mission-critical business requirements. This is a tremendous benefit to the Linux community. It reinforces the value of Linux in the global market and generates renewed interest in applying Linux in place of other more costly operating systems. Solid Server's open platform architecture is making it easier. The informed purchase decision can no longer be based entirely on “which” RDBMS product to buy, but must also consider on which platform it will be used. Given Linux's return on investment, one begins to lean toward “free” Linux. Solid Server 2.2 raises the bar for competing RDBMS producers and demonstrates a strong commitment to Linux as a world-class operating system.

I give Solid Server for Linux v.2.2 a “solid” two thumbs up.

Bradley J. Willson currently designs and troubleshoots tooling for the Boeing 777 program. He owns and operates Willson Consulting Services and in his spare time listens to jazz and plays guitar. He can be reached via e-mail at cpu@ifixcomputers.com.

Archive Index Issue Table of Contents

Advanced search

Beginning Linux Programming

J. Mark Shacklette

Issue #41, September 1997

The book is an achievement both in terms of its subject coverage as well as its surprising depth for a text whose focus, as the title states, is the neophyte.

- Authors: Neil Matthew and Richard Stones
- Publisher: Wrox Press
- Pages: 710
- ISBN: 1-874416-68-0
- Price: $36.95
- Reviewer: Mark Shacklette

Every once in a while a book appears which so perfectly fills a void that its acceptance is all but guaranteed. For Linux programmers or anyone interested in learning Unix programming, there is a new volume to add to your library: *Beginning Linux Programming* by Neil Matthew and Richard Stones. The book is an achievement both in terms of its subject coverage as well as its surprising depth for a text whose focus, as the title states, is the neophyte. The authors' aim is “to cover enough about a wide range of topics to give you a good beginning in each subject.” In this they succeed magnificently. Inside are full chapters on shell programming (**/bin/sh**), an introduction to the Unix file system, terminal I/O, the curses library and a chapter on processes and signals. There is a chapter on how to use the GNU gcc compiler, the gdb debugger, make, RCS and other development tools. (They even teach you the basics of how to write a man page for all those new applications you'll write after mastering Linux programming.) The authors cover Linux's X/Open-compliant dbm database (some distributions have gdbm, the GNU flavor of dbm). They devote one hundred and fourteen pages to interprocess communication alone, including pipes, FIFOs, System V IPC (Semaphores, Message Queues, Shared Memory) and Berkeley sockets. There is a 93-page introduction to X programming which focuses primarily on Tcl, Tk and Wish, along with a sip of

Java. The book concludes with two chapters devoted to Internet programming, one covering HTML and the other CGI.

With such a vast scope, something had to be left out. Coverage of Expect is assigned to two short paragraphs. Python is missing, as is the majority of Motif programming outside of Tcl/Tk (Xlib and Xt are only mentioned in passing). The only coverage of Perl is an example CGI script, and the section on Java could pass by you entirely, if your nose is not careful to pick up the scent of the fresh brew at the close of the discussion of X programming. (Interesting that Java is appended to X programming, not the Internet programming chapters.) Nevertheless, such focus is necessary to avoid a text of several volumes and to concentrate on the omissions would be to miss the point of the work, which is to introduce you to Linux programming through the use of hands-on code examples, which are graduated and presented in a step-by-step fashion. Each new topic is presented in a few short paragraphs and is followed by a succinct but complete “Try It Out” code example.

Over 10,000 lines of source code fill the pages, all of which are available for download (the ubiquitous CD is missing from the back cover). With such a supply of source code, the usual peccadilloes occur, such as a case statement in the section on shell programming, that lists the case as “**[nN]***” (with quotes) instead of **[nN]*** without the quotation marks (the former fails if you enter anything beginning with a **n** or a **N**); or when, during the discussion of pipes, the line:

```
   sprintf(buffer, "Once apon a time, there ..."\n)
```

is magically spell-checked to “Once *upon* a time” in the book's output listing (unfortunately, my compiler isn't as smart as the editors and keeps the spelling as “apon”). All in all, the source code compiles and runs fine. Wrox Press, the publisher, has the source and a sample chapter on its web site for your perusal, at http://www.wrox.com/.

The example code gets to the point. The authors present the minimum amount of code to illustrate an idea which, once given, is quickly summarized as they move on to the next extrapolation or improvement. This approach allows the new Linux programmer to get a firm grasp on the subject without too much diversion or interdependence. All examples are self-contained and may be compiled and run—there are no code fragments here. Such an approach facilitates and encourages the reader to experiment with the code examples, since they are so confined few side effects can be introduced, even by the most intrepid of newbies.

Although most of the examples are focused, the authors do present and develop a larger application that implements a simple audio CD cataloging

program. The application begins life early in Chapter 2 as an extremely simple shell script that exercises the use of flow-control statements in shell programming and uses simple text files for a database. The application reappears in Chapter 6. This time rewritten in C with the same functionality as the shell script version and with a new interface designed to exercise the curses terminal library. In Chapter 7, the application acquires a real database, dbm. In Chapter 11, the application evolves into a small client-server system, which separates the database from the user interface through the use of named pipes. This example illustrates process synchronization and bi-directional data flow between a single server and multiple clients. In Chapter 12, the application substitutes a message queue for the named pipes from the previous chapter, which solves some problems with I/O synchronization previously encountered with named pipes. Finally, the CD application takes to the Web in Chapter 17 as an example of an HTML interface talking out a CGI layer to the database server. This chameleon-like application greatly enhances the communication of many of the principles presented in the book.

In short, *Beginning Linux Programming* is a tutorial on the major topics in Linux programming. If you are willing to spend some time getting up to speed, you will find yourself with a book that not only will hold your hand in the midst of the Magic Garden, but will eventually provide you with a fine pair of walking shoes, well preparing you for your own explorations. With this book at your side, as my son's beginning reader puts it:

You have brains in your head.You have feet in your shoes.You can steer yourselfany direction you choose.

Mark Shacklette is a Principal with Lake Shore MicroTech Group, a Chicago-based consultancy specializing in client-server development in Unix and Windows NT. When he's not with a client, he works on his Ph.D. in the “Committee on Social Thought” at the University of Chicago. He lives in Des Plaines, Illinois with his wife Karen, son David and two cats. He can be reached at jmshackl@midway.uchicago.edu.

Archive Index Issue Table of Contents

Advanced search

Linux Configuration and Installation, Second Edition

Harvey Friedman

Issue #41, September 1997

Linux Configuration and Installation is a book/2-CD package that includes the Linux 2.0.0 kernel, the Slackware 96 distribution, numerous games, utilities and programs.

- Author: Patrick Volkerding, Kevin Reichard, Eric F. Johnson
- Publisher: MIS: Press
- Pages: 522
- Price: $39.95
- ISBN: 1-55828-492-3
- Reviewer: Harvey Friedman

The obvious question that a potentially interested reader might have when seeing a second edition of a useful book is “What are the changes and have they improved on the first edition?”

Linux Configuration and Installation is a book/2-CD package that includes the Linux 2.0.0 kernel, the Slackware 96 distribution, numerous games, utilities and programs.

The general outline of the book is the same as in the first edition; that is, it includes sections on “Linux Installation & Configuration”, “Using Linux”, “Linux Communications and Networking” and “Linux Programming”. However, the chapters within these sections have been revised extensively, both in order and content.

The first section, “Linux Installation & Configuration”, contains three chapters. Chapter 1, Linux and PC Hardware, is roughly the same with some newer hardware covered. Chapter 2, Installing Linux, is largely revised with an emphasis on an MS-DOS or Windows-based installation. A new feature is a section on starting Linux from Windows 95, but since I refuse to use Windows

95, I can't comment on whether this option works. There is also a section on upgrading from a previous version of Linux. In essence, the author recommends removing the old version entirely, particularly if you are going from a.out to ELF. I did this but failed to realize that the new version took much more space for the same packages; thus, my 120MB partition filled before the packages had all been installed. I had to repartition my disk before the new version would install properly. I think it would have been helpful to have the expanded size of all the Slackware disk sets listed, so that partition size could be better estimated. Chapter 3, Installing and Configuring XFree86, does a fairly good job of explaining X. The text describes in detail how to use xf86config without indicating its location; a less experienced user would probably not know to look for it in the /usr/x11/bin directory. This chapter was both 3 and 4 in the first edition.

The second section of the book, “Using Linux”, contains chapters 4 through 6. Chapter 4, Basic Linux Tools, is pretty much the same as Chapter 5 of the first edition.. Chapter 5, Linux Applications, is an expanded version of the first part of Chapter 7 from the first edition. Included are the introduction to Ghostscript that Steve Wegener asked for in his review of the first edition that appeared *LJ* (Issue 23, March 1996), a discussion of Mtools for MS-DOS file systems, some X applications and emulators for various older computers including DOSEMU 0.60.4. Chapter 6, Basic Linux System Administration, expands on the material in the last part of Chapter 7 from the first edition. It is well written and draws upon other Unix writings of Reichard and Johnson.

Section 3, “Linux Communications and Networking”, contains Chapters 7 through 9. Chapter 7, Linux and Telecommunications, was part of chapter 8 in the first edition and deals with serial communications using seyon, minicom, xminicom and rzsz. It is a short, 13-page chapter. Chapter 8, Linux Networking, is an even shorter 5-page chapter covering TCP/IP. It is assumed that the computer is directly cabled to the network via an Ethernet card. Chapter 9, Linux and the Internet, covers dial-up IP connections, electronic mail, the World Wide Web and web browsers, UUCP, FTP, telnet and Usenet newsgroups. I think that the discussion of dial-up IP would have fit better into Chapter 7.

Finally we have Section 4, “Linux Programming”, which consists of one chapter, Chapter 10, Programming in Linux. This appears to be the same Chapter 10 from the first edition. It is replete with short examples and simple explanations for many tools, including cc, make, LessTif, Tcl/Tk, Perl, gawk, etc.

There are two CD-ROMs included with the book, but the page describing the contents of each is missing quite a bit. There is almost no description (it stops after a few words of a sentence) of what is on the first CD (it's a fairly standard

Slackware 96); however, there is a decent description of the second. The directory of the first one is shown in Listing 1.

To quote the page for the second one, “The second CD-ROM contains useful source code (and in some cases, precompiled binaries) for Linux/UNIX applications and utilities mentioned in the book, as well as selected archives from sunsite and tsx-11”. Some of the more interesting programs included, in the order of the page listing, are diald, slirp, several email handlers, WINE and NTFS, POV-ray, several multimedia and/or image processing programs, networking packages including Apache and Samba, office packages, LessTif, Mesa, Java, Perl-5.002, applications, xwatch and other window-managers.

This book/CD-ROM combination is a definite improvement over the first edition, offering more information and better explanations. One further addition that I think would improve the product considerably relates to a sentence on page 243: “As a matter of fact, when you installed Linux, you unwittingly set up dozens of linked files...”--unwittingly is problematic. Having a list of all the links, particularly non-standard ones belonging to other Unices, and a list of important files in non-standard locations would make it much easier for experienced Unix users to recommend Slackware as highly as other distributions.

All in all though, if one intends to use Slackware 96, particularly with no previous Unix or Linux experience, this book/CD-ROM is the one to buy.

Harvey Friedman is a computer consultant at the University of Washington, functioning either as system administrator or statistical analyst. Currently his work requires data analysis using SAS on large datasets. He doesn't spend as much of his leisure time as he'd like playing with Linux, because orienteering, the sport of navigation, is so much fun. He feels going from staring at a computer screen to moving oneself through the forest is a great way to retain sanity. He can be reached via e-mail at fnharvey@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

Building an ISP Using Linux and an Intranet

Eric Harlow

Issue #41, September 1997

This article describes how you can start an ISP and/or create an Intranet using Linux and a dedicated 28.8 connection.

I became the founder of a small ISP for the Baltimore, Maryland area by accident. My wife and I had been having trouble finding a cheap and reliable Internet access provider. After endless frustration with busy signals, we decided to get a dedicated 28.8 line and use it as our connection to the Internet. The connection would always be up and available, and we could both use it to surf the Internet. It seemed natural to take the next step and provide access to others. Beyond helping people get on the Internet and giving me technical experience with Linux, it could provide us with a source of revenue to recoup some of the costs. Before you decide to become an ISP and make lots of money, however, I should warn you competition is fierce and we are not yet profitable.

The Internet server setup we have is a simple one: a Linux machine with four incoming lines for dialup and one outgoing line to maintain the dedicated connection. We currently support about twenty users on the machine. Although simple, this setup could present a problem: if four users dialed in, each would receive (28.8/4) = 7.2Kbps for his or her connection. Most of the time, however, only one user is logged onto our machine, and he receives full bandwidth. In addition, when multiple users are logged on, one may be downloading or composing e-mail, another might be reading a web page and a third may have walked to the bathroom.

One of the first steps you need to take to build a Linux Internet server is to recompile the kernel with networking on, IP forwarding/gatewaying on and any additional drivers selected. We have a network card and a Cyclades Cyclom 16YeP card (a multiport serial card) on the Linux machines, so we have those options turned on. When you turn on the IP forwarding/gatewaying, it enables

the Linux machine to forward packets it receives over your network to the Internet.

Another important step during setup is to get an account with your ISP that supports point-to-point protocol (PPP). You can get a dedicated 28.8 connection for about $100/month that will provide you with full access to the Internet for your network. It is also preferable that you sign up for a static IP address or a Class C address and a domain name. A static IP address lets you log on to the Internet with the same IP address all the time. We have a Class C address, so this discussion uses only the Class C address. A class C address gives you a block of addresses (255) you can use to set up your internal network (Intranet). If you don't have a Class C address but rather a static IP address, you'll have to use IP Masquerading to access the Internet from your LAN. [See "IP Masquerading with Linux" by Chris Kostick in the July 1996 issue of *Linux Journal*—Ed.]

We started building our Internet server by calling our ISP and signing up for a Class C address with a domain name and a dedicated 28.8 line. After we received our Class C address block (206.156.217.*), I picked one of the addresses for our host (206.156.217.10) and proceeded to set up our network using the **netconfig** utility bundled with Linux.

Your ISP should be able to maintain the Domain Name Server (DNS) entry for your machine. The DNS entry allows Internet surfers to access your IP address using your domain name. IP addresses like ours (206.156.217.10) are hard to remember and non-descriptive; the DNS entry allows surfers and customers to reach our machine using NetBrain.com without knowing the IP address.

In the resolv.conf file, there should be a listing of the domain name servers you'll be accessing. The file should look something like:

```
#resolv.conf
Nameserver 206.156.208.2
nameserver 206.156.208.8
```

Setting up PPP

Using the **ppp-on** script (part of the **pppd** package), you establish how your want your PPP connection set up. The **ppp-on** script is shown in Listing 1. You use this list to set up parameters such as the IP address of your machine and the host machine, whether you're running pppd on a modem or through a network, and the device you're using to make the connection. The **ppp-on** script calls the **ppp-on-dialer** script which actually dials the modem to connect to your ISP. The **ppp-on-dialer** uses **chat**, which dials the modem and also handles getting past the ISP's startup screen (user name, password). (Passing the 0.0.0.0

as a parameter for the remote is another way of saying “we don't care.”) The important parameters in the **ppp-on** script include:

- 115200—serial port speed—this doesn't mean your modem is as fast, but with some compression, it might get close.
- /dev/ttyC2—your modem'd port default route—specifies this connection is the default route out of the machine, if it can't find the address locally.

Listing 2. ppp-on-dialer Script

Note that the chat has the **ogin my_login** and **assword: my_password** lines. This is chat's way of saying, “If you see **ogin:** then type **my_login**; then when you see **assword** type **my_password**.” You'll have to dial in manually to your ISP using a terminal program to see how this login screen looks.

If you type **ppp-on** and hear the modem dial and connect, you've taken your first step to running an Internet server. When PPP is running, you should be able to **ping** one of your ISP's machines from the Linux shell. A good address to **ping** is your ISP's DNS machine.

If **ping** is successful, try to see if TELNET works. A simple check is to **telnet** to one of the MOO sites (**telnet baymoo.org 8888**). If that works, you are connected and being routed correctly. If you have an account on another machine, you can also test the incoming connection via TELNET. This is more a test to make sure your ISP has the correct DNS entry. If you get the **Unknown host** error, either your ISP didn't put the entry in or it hasn't made it out—sometimes it takes a few days to make it to all the other machines.

Connecting Other Machines to Your Server

Once your connection to the Internet is stable, it is time to connect your network. Your Linux machine and your other machines should all have network cards installed, and your Linux machine should have the kernel compiled with the appropriate drivers.

Suppose you want to set up Doofus (a Windows 95 client) and hook into your network to give it access to the Internet. Pick an IP address for Doofus of 206.156.217.7 (it can be any number available within your Class C block). To set up the Windows 95 machine to access your Linux server, you must go into the **Control Panel** and pick **Network**. Make sure TCP/IP is bound to your network card. The Properties button lets you set up the following items on each of your machines:

- IP Address: If you have a Class C address, you can assign an IP address 206.156.217.7 and a subnet mask of 255.255.255.0.

- DNS Configuration: Pick **Enable DNS** and give your machine the name **Doofus** for the host and **NetBrain.com** for the domain. This setup provides the Windows machine with the name **Doofus.NetBrain.com**—or read another way, **Doofus** is within **NetBrain.com**. The DNS Server Search Order should have your DNS entries added to be the same as the Linux server's /etc/resolv.conf nameserver entries.
- WINS Configuration: Pick **Disable WINS Resolution**
- Bindings: Add a gateway to your Linux machine. The gateway helps your machine find its way out of the network and onto the Internet. We added 206.156.217.10 to the list of installed gateways.

Test Your Windows 95 Client

After you've rebooted your Win95 machine, you should be able to **ping** it from your Linux host using **ping 206.156.217.7**. If it fails, possible problems could be the cable, Linux drivers, Win95 drivers or your Win95 configuration. Now from the other side... When you're ready to test your Windows 95 client, open an MS-DOS window and ping your server. The command **ping 206.156.217.10** should get a response from your server. You should be able to TELNET to your machine (**telnet mickeymouse.com**) and should also be able to bring up a browser and go to any web site that interests you. It's that easy.

Multiport Serial Card for Dial-up Access

Most personal computers have only two serial ports, and one of those is usually used by the mouse. The best way to provide dial-up access is to purchase a multiport serial card. We use the Cyclades Cyclom 16Yep card, which provides 16 serial ports for modem use. More important, the drivers are built into the Linux kernel.

Before you purchase a specific card, make sure the drivers for the card exist and your machine has the drivers compiled into the kernel. You might have to create the ports your serial card uses with **MAKEDEV**. Our Cyclades card uses ttyC0-ttyC15 for the serial ports instead of the standard ttyS0 and ttyS1 for the standard serial ports. Fortunately, the Cyclades card came with a **makecyc** install script that did the work for me.

Initializing Serial Ports

The program **setserial** needs to be called to initialize the serial port(s). The /etc/rc.d/rc.serial file may need to be edited to properly set up your server's serial ports. To use com2 for the dial-out modem, put the following line in rc.serial:

```
#standard serial port - com2:
setserial /dev/cua1 spd_vhi auto_irq autoconfig
```

For the Cyclades card, I configured the ports 3-10 /dev/cub2 - /dev/cub10 (some unused—for expansion) as follows:

```
#configure Cyclades serial ports
setserial -b /dev/cub2 spd_vhi autoirq skip_test
setserial -b /dev/cub3 spd_vhi autoirq skip_test
setserial -b /dev/cub4 spd_vhi autoirq skip_test
setserial -b /dev/cub5 spd_vhi autoirq skip_test
setserial -b /dev/cub6 spd_vhi autoirq skip_test
setserial -b /dev/cub7 spd_vhi autoirq skip_test
setserial -b /dev/cub8 spd_vhi autoirq skip_test
setserial -b /dev/cub9 spd_vhi autoirq skip_test
```

Make sure the rc.serial file is called from one of the startup rc files, usually rc.s. This will configure your serial ports automatically during boot.

Modifying the gettys File

Next, you need to configure the host to listen to the serial port for incoming connections and to to answer these connections. The /etc/gettydefs file is used to set up the **gettys** which make connections to the machine. When a standard version of Linux is installed, you find these lines in the /etc/gettydefs file:

```
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
c2:1235:respawn:/sbin/agetty 38400 tty2 linux
c3:1235:respawn:/sbin/agetty 38400 tty3 linux
c4:1235:respawn:/sbin/agetty 38400 tty4 linux
c5:1235:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux
```

This provides your console (keyboard) with six virtual logins. The fourth item in the line **/sbin/agetty** is the program polling the console for a login. The following parameters describe the login speed, terminal number and terminal emulation. You add the following lines for dial-up lines after the parameters list.

```
# Dial-up lines using /sbin/getty
# (actually getty_ps)
s1:345:respawn:/sbin/getty ttyC2 115200 vt100
s2:345:respawn:/sbin/getty ttyC3 115200 vt100
s3:345:respawn:/sbin/getty ttyC4 115200 vt100
```

We use a different getty (getty_ps) for our dial-up lines because of trouble using agetty on the serial port. We also also heard that getty_ps is more reliable. You can also use mgetty for the dial-up lines, but getty_ps works great for us. The parameters for getty_ps are slightly different, however: parameters following the getty name are the tty, the /etc/gettydef label and the terminal emulation default. The **115200** in the preceding lines refers to the label in /etc/gettydefs file shown here:

```
#/etc/gettydefs
# Modem locked at 115200: Serial port is at
# 115200, modem is much less, but should be
# able to compress.
#
# Last line of this file is described in next
# comment line as fields separated by # signs.
```

```
# label # initial-flags # final-flags # login prompt # next label
115200# B115200 CS8 CRTSCTS # B115200 SANE -ISTRIP CRTSCTS #@S login: #115200
```

Now you have to provide the getty_ps with the startup values. In the directory /etc/defaults, place the configuration files for each dial-up line. For the dial-up line /dev/ttyC2, we have a corresponding file called /etc/default/getty.ttyC2 shown in Listing 3.

If everything works as planned, the host should be able accept shell logins. You should be able to dial into your machine and run commands in the shell.

To monitor the dial-up connection, you can set the **DEBUG=777** in the /etc/default/getty.tty?? file to create a log file. This will help you identify problems should the modem not answer or not configure properly. The output is dumped to the syslog file usually in /var/adm/syslog.

Dial-up Shell Access for Users

This confirmation, which provides people with shell dial-up access, can be modified to provide dial-up PPP access to customers. We chose to modify the default **login** program (in the **poeigl** package) because we wanted to provide both PPP and shell access (useful when I'm remotely setting up someone's machine). The **ppplogin** program has a prompt that looks like this:

```
Username: jsmith
Password:
Please select PPP or Shell access:
1) PPP
2) Shell
Please enter your choice: 1
```

If the user picks the **shell**, Linux invokes the standard defined shell for the user. If PPP is selected, a script invokes **pppd** for the dial-up user and dynamically allocates him an IP address. Part of the C code for invoking the ppp script file looks like this:

```
/* --- PPP account login --- */
execlp ("/bin/sh", "-sh", "-c",
   "/etc/ppp/ppplogin", (char *)0);
fprintf (stderr,
   "login: couldn't exec shell script: %s.\n",
strerror (errno));
exit(0);
```

The /etc/ppp/ppplogin is shown in Listing 4.

When a user selects ppp, the server looks up the tty the person dialing in is using and assigns the tty an IP address. If the user always calls in on a specific line, he is given the same IP address. A user dialing in on the first line comes in on ttyC6. This is used to assign an IP address of 206.156.217.31 to the user. This creates a PPP link to the dial-up line like the PPP link to my host. The important

parameters related to this tty/ppp connection in the ppplogin script are as follows:

- Detach—don't run as a background process.
- Modem—use the carrier lines to detect things like **hanging up**
- 206.156.217.10:206.156.217.35—I am known as 206.156.217.10, and the person on the other end is known as 206.156.217.35.

Keeping the Connection Established

Early on we found our dedicated connection was frequently dropped by the phone company. I solved this problem by using a program called **pppupd** which constantly pings our ISP's machine and, if the **ping** fails, it invokes the **ppp-on** script to redial the connection.

Fax Services and Seamless Windows 95 Dial-up

Most of our customers are Windows 95 users who did not like having to type their name and password in each time they logged on to the server. To remove this source of irritation, we found a different getty package called **mgetty**, which provides autodetection of PPP dialers for Windows 95 users who want to use the Dial-up Networking dialog box. This has saved us quite a bit of time supporting Windows 95 users. The **mgetty** package (http://sunsite.unc.edu/pub/Linux/system/Serial/mgetty+sendfax) is difficult to set up, so read the documentation before building. One wonderful feature of this package is the capability to receive faxes on the incoming modem lines without additional hardware or additional lines. We can use the same dial-up lines to receive faxes.

E-mail

E-mail for us was automatically configured with my Linux installation. You can install **pine** for shell access and POP v3 for POP server e-mail. If you don't have the POP server installed, you can get a package called **pop3d** from any of the various sites, such as sunsite.unc.edu, and follow the instructions to install it.

Web Server

If you want your machine to host web pages, you have to install a web server. We downloaded the Apache web server (http://www.apache.org/) and recompiled and configured it using the available documentation. Compiling the source should create an **httpd** executable which can be copied into /usr/sbin. In addition, add the line **/usr/sbin/httpd** in the /etc/rc.d/rc.local configuration file for it to be automatically started during boot up.

Eric Harlow has been running NetBrain on Linux since February 1996. He's currently a consultant at RDA Consultants Ltd. His e-mail address is brain@netbrain.com.

Archive Index Issue Table of Contents

Advanced search

Speaking SQL

Reuven M. Lerner

Issue #41, September 1997

An introduction to building a database using SQL in Perl and CGI.

In my work, I often find myself writing CGI programs that need to read or write information on the file system attached to a web server. Sometimes, this information is fairly simple, throwaway stuff, such as logging information accrued when I am trying to debug a particularly difficult program.

Sometimes, as we saw in a series of columns earlier this year, we can use text files for the storage and retrieval of structured information, such as the questions and answers for a multiple-choice quiz. Those quizzes were stored in a simple format, with each question placed on a line by itself. For example, here is a line that might have come from one of those quizzes:

```
What color was George Washington's white horse? White    Black    Gray    Pink    a
```

While the mechanics of magazine publishing mean that you cannot see the difference between various whitespace characters, the above line is separated into six fields: The question text, the four answers presented to the user, and a letter (**a, b, c** or **d**) indicating which of the four answers is correct. Fields are separated by Tab characters (ASCII 9), which look identical to space characters (ASCII 32), but which are quite different as far as the computer is concerned.

The quiz programs we explored earlier this year expected to read from files containing one or more such lines, with each line representing a single question. A quiz containing a single question (for users who prefer easy challenges) would have one line, while a quiz containing 1000 questions would contain 1000 lines.

This raises the important issue of scalability, the software's ability to remain efficient even when data sets become quite large. It is not difficult to write programs that can handle small amounts of data efficiently, particularly as

hardware continues to drop in price while increasing in performance. It is much harder, though, to write software that can handle large amounts of data.

ASCII text files are wonderful when dealing with small amounts of data, since they are easy to manipulate from within programs, particularly when using Perl, which is strong in handling regular expressions. But when we must work with a large amount of data, or when we want to perform sophisticated searches, we may find ourselves reinventing the wheel or working with tools (such as ASCII text files) that no longer fit our needs.

Basic SQL

A common solution to this problem is to off-load the data storage and retrieval to a program known as a relational database server. The "server" part of the name indicates that it expects to receive requests from one or more clients, and the "database" part of the name indicates its storage and retrieval of information on behalf of those clients. You may, however, be unfamiliar with the "relational" part of the name, which means that data is stored in sets of tables, which we can access using SQL, the Structured Query Language. This month, we take a first look at SQL queries, including how they can be integrated into our CGI programs; in the coming months, we will explore this topic in greater depth, using relational database servers for a variety of projects.

SQL is an international standard to which many corporate databases adhere. While the "L" in SQL stands for "language", it does not mean that you can write programs in SQL. Rather, SQL is a language for formulating queries to database servers. The SQL commands must be incorporated into programs written in a true programming language, such as Perl or C.

Relational databases work on the client-server model, just as the Web does. Whereas web clients and servers communicate using HTTP, database clients and servers communicate with SQL. Needless to say, SQL is much more complicated than HTTP, although as you will see, it is fairly straightforward to learn. SQL may be easy to learn, but that does not mean it is simple. On the contrary, long-time database administrators and programmers understand more about the storage and retrieval of data using SQL than I could ever imagine.

The key to understanding SQL is to realize that everything in an SQL database is stored in a table. Rows in the table represent table records, while columns represent fields. Thus, we could represent an address book as a table.

```
Name    Telephone
----    ---------
Reuven  04-824-2265
Andy    02-123-4567
Gil     04-999-8888
```

There are three records in this table, each represented by a row. Each record contains two fields, each represented by a column. Each table and column must have a name, so we will call this the “phone_book” table, with two columns, “name” and “telephone”.

So far, this might not seem like a great advance over what we have done with text files. Why bother with rows, columns, and tables when we can use an ASCII file?

The simple answer is that we can allow the database server to do the work for us—and it will return an answer to us very quickly, as per our instructions, without getting bogged down by the number of records in the database. If we were interested in finding Andy's telephone number with a text file version of the above table, we would need to iterate through the entire file, checking each record for a match. With a relational database, we can issue an SQL query to the database server, asking for only those rows which match our particular criteria.

Thus, if we were interested in retrieving Andy's telephone number from the table above, we could use an SQL **select** command to do so:

```
select telephone from phone_book where name =
        "Andy";
```

The SQL statement above asks the database server to return the **telephone** column from the table named **phone_book**, for each row in which the name is Andy. If a single row matches the query, we receive a single row as a response from the database server, but if multiple rows match, we receive all of those rows. If no row matches our query, we receive no rows, which might seem odd, until you realize that database client programs often iterate over the results returned to them. Iterating over no values is as easy as iterating over 100 values, although most good client programs check to make sure that at least one row was returned.

We can insert a row into our table with the following:

```
insert into phone_book (name,telephone) values
        ("Iris","04-999-8888");
```

After performing the operation above, our table looks like:

Name Telephone---- ---------Reuven 04-824-2265Andy 02-123-4567Gil 04-999-8888Iris 04-999-8888

which we can see by retrieving everything, using an asterisk to mean “all columns”:

```
select * from phone_book;
```

If we want to retrieve all of the rows belonging to people with the telephone number 04-999-8888, we use this line:

```
select name from phone_book where telephone =
        "04-999-8888";
```

Note that we do not need to worry about two identical records, since relational databases strictly require each record to be unique in some way. Two rows might differ in only a single column, but that column is enough to make the rows distinct.

One advantage, then, of using SQL and a relational database server is the increased efficiency, both of our programs (which no longer need to read the entire contents of a text file) and ourselves (since we no longer need to write matching engines and define data formats). There are other advantages to using SQL and relational databases too; most importantly, database servers handle file locking in a sophisticated and efficient way, ensuring that data is not lost while keeping operations moving along quickly.

Relational databases also offer an amazing array of optimization techniques and security levels, among other things. And best of all, SQL is a portable standard that (mostly) works in the same way on a great many database systems; that is, once you learn how to write some basic SQL queries, you will be able to store your data on just about any available platform.

Most of my SQL experience is with Sybase on Solaris systems, but for the purposes of this article, I decided the time had come to install a relational database server on my Linux machine (running Red Hat 4.0 with a number of updated packages, including the 2.0.30 kernel). I decided to download MySQL, a database server that looked small but powerful, and which came in RPM (Red Hat Package Manager) format, allowing me to install it quickly. (Don't confuse MySQL with mSQL, another relational database package available for Linux. For information on how to obtain MySQL, see the sidebar accompanying this article.)

Using SQL from Perl

MySQL comes with a client program named, oddly enough, **mysql**, which allows us to enter SQL queries directly to the database server, which is presumably running at all times. We enter the database with:

```
[1016] ~% mysql test
Welcome to the mysql monitor. Commands end with ; or \g.
Type 'help' for help.
mysql>
```

Just as file systems store files within subdirectories within directories, relational databases store tables inside of databases inside of the overall structure. Thus, when we enter MySQL, we need to specify the name of the database we would like to use. In example above, we specified the **test** database, to which all users have access without needing to go through the standard procedure of entering a user name and password. While user names and passwords for relational databases can be the same as those for the user's account on the system, they do not need to be. Indeed, for the sake of system security, you should make them distinct from your regular system passwords.

Generally speaking, it is also a good idea to create one or more databases exclusively for CGI programs, in order to avoid giving programs complete access to all databases on the system. The nature of CGI programming is such that users might be able to read the user name and password from the program's source code, thus giving them access to whatever tables are in a given database. However, in the interest of time and space, I encourage you to read the MySQL documentation, which describes how to set user permissions for various databases on the system In the meantime, we will use the **test** database, to which all users have access, for our examples

To create our telephone directory table, we type:

```
mysql> create table phone_book (name char(255),
        telephone char(255));
```

Whitespace is unimportant in SQL queries. In the above example, I pressed enter between the end of the first line and the **go** statement on the second line. As you might expect, the **go** command tells a database client to send the query to the database server, where it is evaluated and executed. Alternatively, we can use a semicolon at the end of our query, which will preclude the need for **go**.

The server responds to our query by giving us some statistics:

```
Query OK, 0 rows affected (0.27 sec)
```

In other words, creating a table took .27 seconds and did not affect any existing rows.

You can quit **mysql** by typing **quit** at the **mysql>** prompt.

The MySQL programmatic interface from Perl works in much the same way as the command-line program, except that it uses Perl 5 objects. The basic idea is straightforward; we create an instance of a MySQL object, and then use that object to get through the process of logging in, sending queries, and interpreting the results.

Listing 1 contains a functional program that can query our **phone_book** table and return the results. More importantly, though, that program is the skeleton for every program we write using MySQL. While the syntax might be slightly different for Sybase and other databases, the general idea is the same—connect to the database server, choose a database, send a query in SQL and iterate through whatever results are returned.

First, we connect to the database server using Unix sockets, in part because MySQL enables those sockets by default, which makes for an easier explanation in a short column such as this one. You can, of course, also connect to a database server running elsewhere on the network, just as a web browser can connect to a web server across a network.

Once we are connected to the MySQL server, we use the query method to enter our SQL query. Just as connecting to the database returns the database handle **$dbh**, sending an SQL query returns the statement handle **$sth**. And just as we need to use **$dbh** in order to send a statement, we need to use **$sth** in order to retrieve results. In this particular statement, we have asked to see both of the table's columns, as well as all of the rows in the table. However, we could restrict our query with a **where** clause, as described earlier, which would return a subset of the table's rows. We could also ask for a subset of the table's columns, such that only the name or the telephone number would be returned.

Results are retrieved by iterating over the rows that were returned from the server. If no rows match our query, the iteration is not performed; if 100 rows match our query, it is performed 100 times. If we are interested in maximizing the efficiency of our programs that handle SQL queries, it is in our interest to construct queries that return only those rows that most interest us, since iterating through a large number of rows can be quite inefficient and time-consuming.

If I run the program in Listing 1 (named sql-test.pl on my system) from the command line, I get:

```
[1031] ~/Text/LJ% ./sql-test.pl
Iris    04-999-8888
Reuven  04-824-2265
Andy    02-123-4567
Gil     04-999-8888
```

We can, of course, use the above skeleton program to insert rows, create tables and do more complicated things, such as joining tables together (which is, to a large degree, the magic behind SQL) and order results in ascending or descending order. If we were to keep the area code in a different column from the telephone number itself, we could refine our searches even further, asking for all people within a given area code whose first name is **Iris**, for example.

Using MySQL from a CGI program

Now that we have seen some basic uses of MySQL from within Perl, let's spend some time thinking about how we can integrate the use of MySQL into a CGI program. While this might seem like overkill for some small jobs, database servers are so much more reliable and efficient at this sort of task than our CGI programs that it is almost always worth using such a server, assuming one is available.

By using a database server, we can be sure that our data is stored more reliably than with text files. As an added bonus, the information is available using SQL, which is more efficient and flexible than text files.

How can we use a database server from within our CGI programs? The simple answer is that it is actually no different from connecting to a database server from within non-CGI programs. We still create the Mysql object, use its methods to send an SQL query and retrieve results. The differences are in our ability to modify our query based on input sent to us in an HTML form and the necessity of sending our output to the user's browser using a recognized content type (usually HTML). Such a program, which I have called cgi-sql-test.pl is shown in Listing 2.

While cgi-sql-test.pl is longer than the program on which it is based, it is not much more complicated.

First, we fire up the CGI module for Perl, which you can get via the Comprehensive Perl Archive Network (CPAN) at http://www.perl.com/CPAN. After creating an instance of CGI, we send an HTTP **Content-type** header to the user's browser indicating that we will be returning results of type **text/html**, i.e., HTML-formatted text.

Following our initialization of the CGI environment, we go ahead with what we had done in the non-CGI version of the program, namely connecting to the database, sending our query and retrieving the results.

This is where the big difference lies. Rather than printing the results to standard output, we send them in HTML format to the user's browser, so that we can use all sorts of nifty HTML formatting techniques to display the results.

In this particular example, I decided to put the results of the telephone list in an HTML table, which is attractive and makes it easy to understand the results. The **<tr>** tag introduces a table row, while the **<td>** tag introduces a column within a row. Because each iteration through the **while** loop represents a new record in the database, we can start a new HTML row at the top of each loop, ending it at the bottom of each loop.

We will continue to explore the interaction between SQL and CGI in the next few installments, but before I conclude this month's column, I want to show at least one example of how we can modify the SQL queries based on the user's input. For the sake of simplicity, we modify our program such that it will ask the database server to return only those rows whose **name** column matches what we enter in the query string. Thus, if we are interested in finding out Gil's telephone number, we can go to:

```
/cgi-bin/cgi-sql-test.pl?Gil
```

And if we are interested in finding out Andy's telephone number, we can go to:

```
/cgi-bin/cgi-sql-test.pl?Andy
```

which produces only that listing.

But what happens if someone invokes our program without entering a name in the query string? Well, our program cleverly notices it and produces a very small page of HTML in response. This small page of HTML asks the user to enter a name for which to search and then uses the **<isindex>** tag to create a text field in the page of HTML.

The **<isindex>** tag has generally fallen out of favor, since HTML forms are more flexible and useful. When a user enters information into an **<isindex>** field and presses enter, the URL in which the **<isindex>** tag appeared is reloaded—with the user's input appended as part of the query string.

Thus, if our program receives no input in the query string, it produces a page containing **<isindex>**. Whatever the user enters in that text field causes our program to be reloaded, this time with a value in the query string. That value is picked up by our program and passed to MySQL, which returns the results in an HTML table.

That concludes the basic introduction regarding the integration of SQL and CGI programs. As you might imagine, SQL databases are far more powerful than the programs and databases we have seen this month. Over the next few months, we will spend some more time looking at different ways in which we can use MySQL (and relational database servers in general) to make for more interesting, efficient and useful web sites.

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.

Letters to the Editor

Various

Issue #41, September 1997

Readers sound off.

Misspelling

It's great to see two (count them) of our books reviewed in a single issue. [June, 1997] However, you spelled my name wrong in the “Programming with GNU Software” article by Randyl Britten. I take that as a lapse on my part. I should be in touch with you more often. (I tend to just drop by the booth at occasional conferences.)

—Andy Oram O'Reilly & Associates, Inc .andyo@ora.com

Thanks for being so understanding. Unfortunately—a-hem—I have to take full responsibility for the spelling. It's in my original manuscript and there was no excuse. I offer my apologies.

—Randyl Britten britten@u.washington.edu

More Novice Articles

Thanks for your seamless renewal of my subscription, even though I had decided in favour of the new *UK Linux World* magazine, which has now disappeared after one issue.

LJ continues as well as ever, but I'd like to add my voice to the novice lobby—more stuff for us please. After some easily understood articles on RCS, shell scripting and the like, you've become all technical and clever again. I'd like to see stuff on make and gdb—I've downloaded many applications as source archives only to find the authors assume abilities I don't have. Stuff on the startup and shutdown scripts would also be nice. Anyway, great magazine.

—Bob Smith bob@bank.demon.co.uk

For every opinion, there's an equal and opposite opinion. Read on —Editor

More Technical Articles

I would like to see *Linux Journal* remain a Linux magazine, and **not** move towards more *WWWsmith* articles, as you have done in the last few months.

Also, I would like to see *LJ* become **more** technical, and move away from the novice corner type stuff, which can easily be found in more current form in places such as *Linux Gazette*, newsgroups, etc.

When I say technical, I mean articles like Alessandro Rubini's excellent Kernel Korner columns. Or detailed articles on Perl and shell scripting, hardware ports to alpha or networking (the article on ghosting in June is a nice detailed article). [“Ghosting Onto the Net”, Scott Steadman, June 1997]

I understand that Linux is new in some sense, and you may feel justified in having the “simpler” stuff in there, so I offer my feedback as simply another data point for your consideration.

—Les Schaffer godzilla@futuris.net

We strive to be balanced, offering both a Kernel Korner and a Linux Apprentice column each month —Editor

Keyboard Typos

I very much enjoyed your keyboard article in the June *Linux Journal*. [“Consistent Keyboard Configuration”, John Bunch] The article as published by *LJ* had several typos. I was able to get most everything to work as described. Did not have much luck with the arrow keys in Emacs running in an xterm. I have not been able to determine the cause of this. Something to do with the xterm translations?

—W. Paul Mills wpmills@sound.net

Note that on page 54, six lines are broken, forcing “Arrow” onto the following line. These lines should be joined so that “Arrow” is inside the comment. Also note that the escape sequences are incorrect. The double backslashes should all be single backslashes, so for example, the line for F117 should read:

```
string F117 = "\033\033[A" # Alt-Up Arrow
```

This type of error is present throughout the article. On page 57, the key translation lines have two problems. First, all of the double backslashes should

be changed to single backslashes. Second, the lines were broken improperly. The first seven lines are shown in Listing 1.

Listing 1. Keyboard Corrections

Copy the rest of the lines from the man page for xterm(1). Every line of the translations, except for the last line, should end with either **\n** or ****. Typographical errors here are very serious, because they cause problems without generating any error messages.

Let me know if this helps.

—John F. Bunch bunch@ro.com

Benchmark Table

I have a little big complaint about *LJ* Issue 37, in particular the native PowerPC article [“Native Linux on the PowerPC”, Cort Dougan, May 1997]. There was a performance listing which compared MKLinux, native Linux-PPC and OSF and some Sun operating systems, but the table was typeset all wrong. Being so cryptic it is almost, if not completely, useless since one cannot tell which digits belong to which columns. If you have so many problems getting your magazine printed correctly, you should probably hire better people, like me for instance.

—Ville Voutilainen vjv@stekt.oulu.fi

Think Linux

Hello. My name is Giacomo Maestranzi. I am from Italy and I am a frequent reader of *Linux Journal* because I find it very interesting—especially the last issue (June 1997) which covered Linux use in my region (TRENTINO ALTO ADIGE and the city is BOLZANO). [“Traveling Linux”, Maurizio Cachia] What I just have to say is that I love Linux and everything related to it, and I have created a slogan for it that I would like to see in a future issue. The slogan is:

THINK FREE..... THINK BIG.....THINK LINUX

Thank you again for *LJ*. I apologize for my English, but this is my best.

—Giacomo Maestranzi hoteuropeo@well.it

Push Media

Kudos to Doc Searls for his insight into how the WWW has the “mainstream media/advertisers” scrambling to remain a monopoly. [“Shoveling Push Media”, June 1997]

I have been an active WWW user for about 2 years—started on AOL as a newbie and graduated to a direct ISP. I am not interested in push technology. My senses are offended often enough when I turn on the TV or radio. Push technology will appeal only to the “couch potatoes” of the world who purchase WebTVs to impress their technophobic friends. I say leave TV on the TV.

The beauty of the WWW is that I can control what I see. If I find a site uninteresting, I leave. The other compelling aspect of using the Web is that I can remove the classic “middle man” from my business transactions. When I want to buy something using the Web, I go straight to the person(s) who offers it. I do not have to be offended, goaded or otherwise angered by traditional advertising. This fact scares technologically savvy advertisers. I would be scared too, but that is the reason I write software for a living.

—Jeffery C. Cann jc_cann@ix.netcom.com

Perfect Box

Eric Raymond's article, “Building the Perfect Box” (April, 1997) was quite instructive, but he did not mention one essential component—the keyboard. In my experience, this is one place you should not skimp. A bad keyboard is frustrating to use and may contribute to carpal tunnel syndrome. I like the top-of-the-line IBM keyboards, but you should always try one out before you buy it. When you experiment, you should sit in the same position you use when typing.

By the way, if there is anyone out there who does not have Raymond's book, *The New Hacker's Dictionary*, go out and buy it right now.

—James R. Miller jimxc@jimxc.seanet.com

Wireless Solution

I read the “Linux Means Business” column with interest this month [“Connecting SSC via Wireless Modem”, Liem Bahneman, May 1997], as I have recently switched from an analog modem to a wireless solution for net access to the office from home. I recommend Metricom's Ricochet service (http://www.ricochet.net/) very highly. I consistently receive transfer rates of 2.5 to 3.5KB/s, and it is child's play to use it under Linux; it supports the standard

Hayes AT command set. I've never received a busy signal and establishing a connection is lightning fast compared to an analog modem's handshake.

—Nick Silberstein nick@fusion.com

Wabi Article

I was surprised to see in the Wabi product review by Dwight Johnson (*LJ*, June 1997) a suggestion to **chmod<\!s>666 /dev/fd0**. Giving random users permission to write to your floppy drive is not exactly a good thing to do.

Also, near the end of the article, it was mentioned that one should wait for Wabi 3.0 to drive 24-bit displays. Last time I checked, Caldera's Wabi 2.4c (not yet released) should fix the 24-bit display problem.

As for the “seamless integration of Microsoft Windows with Linux,” I personally find Wabi's handling of the focus most annoying. For Windows tasks that take time to complete, you can easily create havoc by focusing on a Linux window to do something, just to be interrupted in the middle by a “regain of focus” by a Wabi task. For machines with a lot of memory (and therefore the ability to run two X servers), I find running Wabi on a separate X server to be the safest.

(The article also did not cover keyboard remapping; the information on keyboard remapping found on Caldera's web site is not exactly helpful.)

—Ambrose Liac li@acli%interlog.com

Security and Networking

I just received your recent issue of *Linux Journal*, and it was very helpful and informative. [“The SYN Denial of Service”, Douglas Stewart, et al., June 1997] I do have some problems with the SYN denial of service prevention. The source was published in *Phrack* magazine, and also includes the methods of prevention (which were the same as you had discussed in *LJ*). Unfortunately, TCP SYN flooding is only one of many attacks; there is also Project Hades which deals with TCP exploitation, and Project Loki which presents the theory of ICMP_ECHO tunneling. These are just the articles I have read. If you want to stay ahead in the security field, read these articles as they also contain methods of prevention. *Phrack* can be found at http://www.fc.net/phrack. I hope this is of some help.

—Tom McHannes pnmtofte@imperium.net

Progress on Linux

Just read with interest your 1996 article about your company's use of Progress on Linux. [“Sticking with Progress”, Peter Struijk and Lydia Kinata, September 1996]

As an international non-profit organization, we have used SCO Unix for 8 years and have been very pleased with it, except for the growing resources needed to run it, as well as the cost to buy and upgrade it.

At our international meetings in London last month, we decided to do some testing with Linux to see if it works in our environment and if it could be used as a replacement for our office networking systems. Our German office has been running nicely on Linux for several months, but has yet to get the Progress kit for testing.

1. Did you have a shared library from SCO or did you have to buy the license to make running Progress legal?
2. Are you running Progress 7 or 8 now?

We are looking at linking Progress with “static binding” to eliminate the need for the shared libraries, since we have the full development kit.

Progress says it will probably never support Linux directly since there is no standards body to refer to as there is with its commercial counterparts.

—Ron Tenny ron@omusa.om.org

We actually use the free libraries distributed with iBCS: that is, they are available in a separate archive. The only two we need are libc_s and libnsl_s. I believe we still have an (old) SCO license, but we never had a need to use the original libs (although they work fine, too).

We are still running V6, but I've heard reports from the east coast and Canada that V7 and V8 can be run on Linux without (major) problems.

To join our mailing list for Linux Progress users, send e-mail to info@linuxjournal.com with one line in the body containing the word “info”.

—Peter Struijk info@linuxjournal.com

Archive Index Issue Table of Contents

Advanced search

Education

Marjorie Richardson

Issue #41, September 1997

The most reliable and cost- effective way for a school to get on the Internet is to use the Linux operating system with the Apache web server.

Our focus this month is on Education, a subject dear to the hearts of many Americans. Since coming into office, President Clinton has been promising to get all U.S. schools on the Internet, and Congress has appropriated money to help the public schools “get wired”.

The most reliable and cost- effective way for a school to get on the Internet is to use the Linux operating system with the Apache web server. Currently, Apache is used more than any other available server—it just doesn't get the same publicity commercial servers do because it is freely available. That is, no one benefits financially from advertising it.

Our feature article, “Holt Public Schools and Linux”, is about a school system that has done this very thing—networked their computers using Linux and provided students with access to the Internet. Linux has proved the right solution for their school system, and it is the right solution for others, too.

Universities have been proving the reliability and effectiveness of Linux for some time now. For this issue we received five articles from universities, but had room for only three. The other two will be published in future issues.

As all of these articles show, Linux works for students in the classroom and the lab, as well as at home and in space. Students at the University of Colorado used Linux for the hydroponics experiment aboard the space shuttle [“Linux Out of the Real World”, Sebastian Kuzminsky, *Linux Journal*, July 1997]. This month, we have students at that same university using Linux for robotic car races.

Fall Comdex

As I write, Spring Comdex is being held in Atlanta, Georgia, and Fall Comdex is being planned. Comdex Fall '97 will be held in Las Vegas, Nevada, from November 17 through 21. Details can be found on their web site at http://www.comdex.com/. Comdex Fall is the largest industry trade show in North America, with over 2,000 exhibitors and over 200,000 attendees. This year the Linux Pavilion will be larger than ever with many prominent Linux vendors participating. *Linux Journal* will definitely be there to meet you.

The exhibit halls at Comdex are free to pre-registered attendees. A form for pre-registration can be found at their web site. If you wish to get involved with organization or to set up an exhibition booth for yourself in the Linux Pavilion, send e-mail to info@linuxjournal.com.

Reader's Choice

We are now taking votes for our annual Reader's Choice awards at the *Linux Journal* web site, http://www.ssc.com/lj/. There are more categories this year, so don't miss your chance to vote for your favorite products. Voting ends August 22, 1997.

Buyer's Guide

Our first annual *Linux Journal Buyer's Guide*, which came out in February of this year, has been declared a success, so we are doing it again. This second issue will again be published as a free thirteenth issue for our subscribers. So, subscribe now by e-mailing info@linuxjournal.com.

Product and service listings come from forms that we distribute, as well from the Linux Software Map. If you have not received a form, it can be found on our web site at http://www.ssc.com/lj/bg/. There is no charge to have your product, service or business listed in this issue.

Linux Speaker's Bureau

We have added a new page to our Linux Resources web site, http://www.ssc.com/linux/lsb/. On this page you can find a list of people willing to present talks about Linux. The page gives you some information about these speakers, including talks they've given and talks they'd like to give. When you need a speaker for your club or trade show, all the information is right here. If you like to talk about Linux, there is form for you to fill out that will add your name to the growing list of speakers.

GLUE Announcement

Caldera has announced that it will give a free copy of OpenLinux Lite on CD-ROM for each newly registered group of GLUE. Caldera, Inc. (http://www.caldera.com/) is located in Provo, Utah. For full details on GLUE and to register your group as a member, visit the GLUE web site at http://www.ssc.com/glue/.

Archive Index Issue Table of Contents

Advanced search

Atlanta Linux Showcase Report

Phil Hughes

Todd M. Shrider

Issue #41, September 1997

ALS was a success—everyone had fun and went home happy. Here's a report from Phil, and following it is one from an ALS attendee, Todd Shrider.

The Atlanta Linux Showcase (http://www.ale.org.showcase/) is over, and everyone is beginning to recover. Recover, that is, from being awake too long, being on a plane too long and stuffing more Linux than will fit into one weekend.

ALS was put together by the Atlanta Linux Enthusiasts, the local Linux users group in Atlanta, Georgia. The show began in the evening on Friday, June 6 and ran through Sunday afternoon. More than 500 people attended. The report following this one by Todd Shrider covers much of the show, including the talks.

I want to thank Amy Ayers and Karen Bushaw for making their photos available to us, with a special thank you to Amy for getting them scanned and uploaded to the SSC ftp site. (Additional photos are available on the *Linux Journal* web site in the July issue of *Linux Gazette*, http://www.ssc.com/lg/.)

I spent most of my time in the *Linux Journal* booth giving away magazines and talking to show attendees. One aspect that made this show special for me is the lack of time I spent explaining to attendees that Linux is a Unix-like operating system. Instead, I got to discuss Linux with experienced people with thoughtful questions, letting them know in the process how *LJ* could help them. Each attendee was truly interested in Linux and stopped at each booth in the show. I expect attendees appreciated the low signal-to-noise ratio in the booths; that is, conversations were solely about Linux.

The Roast

On Saturday night there was a roast. No, I didn't change from a vegetarian into a meat eater overnight—we were “roasting” Linus. That is, a group of people presented interesting stories about Linus, intended to only slightly embarrass him.

In front of about 115 people, Eric Raymond, David Miller, Jon “maddog” Hall and I got to pick on this Linus character. Topics varied from Linus almost being hit by a car in Boston, because he was so engrossed in talking about a particular aspect of kernel code, to the evolution of the top-half/bottom-half concept in interrupt handlers and why Linus was apparently moving from geekdom to becoming a “hunk” sportswear model. (See the cover of the San Jose *Metro*, May 8-14, 1997.)

Maddog finished the roasting by telling a few Helsinki stories and showing a video that included Tove's parents talking about Linus. A good time was had by the roasters and the audience, and as Linus's closing comment was “I love you all,” we assume he had a good time, too, and wasn't offended by our gentle ribbing.

The Future

The show came off very well. I consider this success an amazing feat for an all-volunteer effort. The ALE members plan to write an article for *Linux Gazette* about how they made this happen. We'll also make this information available on the GLUE web site (http://www.ssc.com/glue/). I would like to see more shows put on by user groups. The local involvement, the enthusiasm of the attendees and the all-Linux flavor of the show made this weekend a great experience. We are already thinking about a Seattle or Portland show, and we would like to help others make regional shows a reality.

More on ALS

by Todd M. Shrider

I first started writing this article in my hotel room late Sunday evening (or early Monday morning) planning to get just enough sleep that I would wake up in time to catch my plane. The plan didn't work—I missed my 6:00 AM flight out of Atlanta. I did the second draft while waiting for my new 9:45 AM flight. The third draft came (yes, you guessed it) while waiting for my 1:30 PM connection from Detroit to Dayton, also having missed the previous connection because of my first flight's late arrival. Suffice it to say, I'm now back home in Indiana and still enjoying the high I got from the Atlanta Linux Showcase.

Thanks to all the sponsors and to our host, Atlanta Linux Enthusiasts, the conference started with a bang and went off without a hitch. The conference was a three-day event, starting with registration on Friday and ending on Sunday with a kernel-hacking session led by none other than Linus himself. In between, there were numerous conferences found in both business and technical tracks, several “birds of a feather” (BoF) sessions and a floor show. These events were broken up with frequent trips to local pubs and very little sleep.

This was my first Linux conference, and I found that an added benefit of ALS was meeting all the people who use Linux as a business platform and tool. (These same people tend to be doing very cool things with Linux on the side.) From companies such as Red Hat, Caldera, MessageNet, Cyclades, DCG Computers and others, it was obvious that many people have very creative ways to make money with Linux. This enterprising wasn't limited, by any means, to the vendors. Many of the conference speakers spoke of ways to make money with Linux or of their experiences with Linux in a professional environment.

All of these efforts seemed to compliment the key-note address, “World Domination 101”, where Linus Torvalds called for applications, applications, applications. (Did I say he thought Linux needed a few more useful applications?) Anyway, he pointed out the more or less obvious fact that if Linux is going to be a success in a world of commercial operating systems, it needs every application type available for commercial operating systems. In other words, if you're thinking about writing application software for Linux, don't think—just do it. Another thing pointed out by Linux, and which I was glad to hear echoed throughout the conference, is the need for Linux to be easy to use. It needs to be so easy that a secretary or corporate executive could use it as productively as they would Windows 95. We need to make people realize that Linux has eliminated the high learning curve usually associated with Unix.

Don Rosenberg, while speaking on the “how-to” and “what's needed next” of commercial Linux, said that we are now in a stage where the innovators (that's us) and the early adopters (that's us, as well as the people using Linux in the business world today) must continue to push forward so that we can get another group of early adopters (the old DOS users) to take us seriously. In Maddog's closing remarks, he urged us all to find two DOS users, convert them to Linux, and then tell them to do the same. Today, as a step in this direction, I introduced a local corporate computer sales firm to Linux; whether they take my advice remains to be seen, but believe me, I'm pushing.

The rest of the conference was filled with business and technical talks. The business track included such talks as Eric Raymond's “The Cathedral and the

Bazaar”, talks on OpenLinux by both Jeff Farnsworth and Steve Webb, and “Linux Connectivity for Humans” by none other than Phil Hughes. Lloyd Brodsky was on hand to speak about “Intranet Support of Collaborative Planning”, while Lester Hightower brought us the story of PCC and their efforts to bring Linux to the business world. Mark Bolzern spoke of the significance of Linux, and Bob Young talked of the “process”--not the “product”--of Linux.

The technical track started with Richard Henderson's discussion of the shared libraries and their function across several architectures. Michael Maher gave a how-to of Red Hat's RPM package management system, and Jim Paradis discussed EM86 and what remains to be done so that one can run Intel/Linux binaries under Alpha/Linux. David Miller then followed, giving a boost of enthusiasm with his discussion of the tasks involved in porting Linux to SPARC, and Miguel de Icaza took us on a trip to the world of RAID and Linux. We convened the next day to hear David Mandelstam discuss issues involved with wide-area networks, and to hear Mike Warfield's anatomy of a cracker's intrusion.

All in all, the conference was a huge success. As an improvement for next year, I might suggest more involvement from the vendors (or maybe just more vendors), discounted prices for conference attendees from the vendors on their special Linux wares and a possible tutorial session, like those seen at UseLinux (Anaheim, California, January 1997). Otherwise, a few virtual beers (I owe you, Maddog) and lots of great geek conversation made for one wild weekend.

Phil Hughes is the publisher of *Linux Journal*.

Todd M. Shrider (todds@ontko.com)

Archive Index Issue Table of Contents

Advanced search

Linux Grows Up

Phil Hughes

Issue #41, September 1997

Linux is being recognized as a serious OS with real commercial potential.

Each month we allocate a page for this column. The space is reserved past the regular deadline to give us an extra week to find the right earth-shattering event to report—which sometimes doesn't happen. This month we missed the initial deadline and were left with about a day to find some earth-shattering news and write the column.

Back in the early days of Linux there was a new kernel almost every day, which produced a continuous stream of new topics. Linux has grown up—it's too stable, reliable and routine.

Or is it? I went to the comp.os.linux.announce newsgroup hoping to find an exciting event. I didn't. I read it again. Still no exciting event. Then, I realized I was so busy looking for *one* thing that I had missed an event of more significance than any single post.

Linux is being recognized as a serious OS with real commercial potential. It's not that we haven't had anything commercial posted before, it is that there were so many posted in the last week and the type of information posted. Here is a sample:

- **Process credit cards on your computer**: Credit Card Verification System (CCVS) from HKS, Inc. is a package that gives you a command line interface and GUI to do credit card processing as well as libraries to call from programs.
- **WebMagick Image Web Generator 1.29**: WebMagick is a package which makes putting images on the Web as easy as magic. Or, more specifically, WebMagick builds HTML pages and image maps from a set of image files. Thus, rather than manually building a page using thumbnails and writing HTML so the thumbnails are clickable, WebMagick builds maps consisting

of the thumbnails and writes the HTML. Besides saving you time, WebMagick improves performance by decreasing the number of individual files that make up a clickable page.

- **Linux nominated for a European Software Excellence Award**: These awards are sponsored by Ziff-Davis, Europe's largest computer magazine publisher. The three finalists for the Desktop Environment Award were Microsoft, IBM and Red Hat Software. They said: "... Linux has grown up from being a programming freak's playground to a stable and easy to install operating system. ..."
- **Web-based application development platform**: TalentSoft Web+ 3.0 is a premier web-based application development platform for Unix and Windows. The article states "Web+ is extremely scalable, having been tested successfully on a web site with an average of 2,000,000 hits/day, 40% of which are hits to the Web+ server." Now, the posting didn't say it was a Linux box that handled the 2,000,000 hits per day, but the product is available for Linux—the limitation is the hardware, not the software.
- **VBVM—A Visual Basic 5 Virtual Machine**: This product from Softworks Limited is a portable version of the MS Visual Basic 5 virtual machine. It enables you to take VB5 executables and run them, unmodified, on other platforms. While not an application, it will make it possible for lots of existing applications to run on Linux.
- **Rent-a-dedicated-server for $250**: Vipex Internet Presence rents Linux servers including DNS and an unlimited number of domains for $250/month.
- **Qbib-1.1 bibliography management system**: Herrin Software Development, Inc. had built a bibliography management system based on **qddb**. It features all sorts of import and export options plus searching and report generation.
- **Motif Interface Builder VDX 1.1**: VDX from Bredex GmbH is a GUI-based interactive tool that generates C and C++ source code.
- **Regulus 1.1**: Regulus is a package to manage customer accounts for ISPs and includes customer activity logs and a web interface to access those logs.

This is enough of a sample of what's out there to give you the idea. Being an old Unix hacker, I see this influx of postings as the tool box getting filled with new, fancy tools. For example, using Regulus and CCVS, you can quickly put together an ISP with automated credit-card billing. Use Web+, WebMagick and a Vipex server to build a web site.

Tie all of this together with a post about an article on Linux in the June 1997 issue of *Business Computer World* that concludes that "Linux is very solid,

widely used, and a real potential threat to Microsoft." It also states "Linux is behind in the availability of applications, but is catching up."

Maybe some day STP will cover NT being replaced with Linux.

Phil Hughes is the publisher of *Linux Journal* and can be reached via e-mail at info@linuxjournal.com.

Archive Index Issue Table of Contents

Advanced search

Introduction to Named Pipes

Andy Vaught

Issue #41, September 1997

A very useful Linux feature is named pipes which enable different processes to communicate.

One of the fundamental features that makes Linux and other Unices useful is the “pipe”. Pipes allow separate processes to communicate without having been designed explicitly to work together. This allows tools quite narrow in their function to be combined in complex ways.

A simple example of using a pipe is the command:

```
ls | grep x
```

When bash examines the command line, it finds the vertical bar character **|** that separates the two commands. Bash and other shells run both commands, connecting the output of the first to the input of the second. The **ls** program produces a list of files in the current directory, while the **grep** program reads the output of **ls** and prints only those lines containing the letter **x**.

The above, familiar to most Unix users, is an example of an “unnamed pipe”. The pipe exists only inside the kernel and cannot be accessed by processes that created it, in this case, the bash shell. For those who don't already know, a parent process is the first process started by a program that in turn creates separate child processes that execute the program.

The other sort of pipe is a “named” pipe, which is sometimes called a FIFO. FIFO stands for “First In, First Out” and refers to the property that the order of bytes going in is the same coming out. The “name” of a named pipe is actually a file name within the file system. Pipes are shown by **ls** as any other file with a couple of differences:

```
% ls -l fifo1
prw-r--r--   1 andy  users    0 Jan 22 23:11 fifo1|
```

The **p** in the leftmost column indicates that fifo1 is a pipe. The rest of the permission bits control who can read or write to the pipe just like a regular file. On systems with a modern **ls**, the | character at the end of the file name is another clue, and on Linux systems with the color option enabled, **fifo|** is printed in red by default.

On older Linux systems, named pipes are created by the **mknod** program, usually located in the /etc directory. On more modern systems, **mkfifo** is a standard utility. The **mkfifo** program takes one or more file names as arguments for this task and creates pipes with those names. For example, to create a named pipe with the name **pipe1** give the command:

```
mkfifo pipe
```

The simplest way to show how named pipes work is with an example. Suppose we've created **pipe** as shown above. In one virtual console1, type:

```
ls -l > pipe1
```

and in another type:

```
cat < pipe
```

Voila! The output of the command run on the first console shows up on the second console. Note that the order in which you run the commands doesn't matter.

If you haven't used virtual consoles before, see the article “Keyboards, Consoles and VT Cruising” by John M. Fisk in the November 1996 *Linux Journal*.

If you watch closely, you'll notice that the first command you run appears to hang. This happens because the other end of the pipe is not yet connected, and so the kernel suspends the first process until the second process opens the pipe. In Unix jargon, the process is said to be “blocked”, since it is waiting for something to happen.

One very useful application of named pipes is to allow totally unrelated programs to communicate with each other. For example, a program that services requests of some sort (print files, access a database) could open the pipe for reading. Then, another process could make a request by opening the pipe and writing a command. That is, the “server” can perform a task on behalf of the “client”. Blocking can also happen if the client isn't writing, or the server isn't reading.

Pipe Madness

Create two named pipes, pipe1 and pipe2. Run the commands:

```
echo -n x | cat - pipe1 > pipe2 &
cat <pipe2 > pipe1
```

On screen, it will not appear that anything is happening, but if you run **top** (a command similar to **ps** for showing process status), you'll see that both **cat** programs are running like crazy copying the letter **x** back and forth in an endless loop.

After you press ctrl-C to get out of the loop, you may receive the message "**broken pipe**". This error occurs when a process writing to a pipe when the process reading the pipe closes its end. Since the reader is gone, the data has no place to go. Normally, the writer will finish writing its data and close the pipe. At this point, the reader sees the **EOF** (end of file) and executes the request.

Whether or not the "broken pipe" message is issued depends on events at the exact instant the ctrl-C is pressed. If the second **cat** has just read the **x**, pressing ctrl-C stops the second **cat**, **pipe1** is closed and the first **cat** stops quietly, i.e., without a message. On the other hand, if the second **cat** is waiting for the first to write the **x**, ctrl-C causes **pipe2** to close before the first **cat** can write to it, and the error message is issued. This sort of random behavior is known as a "race condition".

Command Substitution

Bash uses named pipes in a really neat way. Recall that when you enclose a command in parenthesis, the command is actually run in a "subshell"; that is, the shell clones itself and the clone interprets the command(s) within the parenthesis. Since the outer shell is running only a single "command", the output of a complete set of commands can be redirected as a unit. For example, the command:

```
(ls -l; ls -l) >ls.out
```

writes two copies of the current directory listing to the file ls.out.

Command substitution occurs when you put a **<** or **>** in front of the left parenthesis. For instance, typing the command:

```
cat <(ls -l)
```

results in the command **ls -l** executing in a subshell as usual, but redirects the output to a temporary named pipe, which bash creates, names and later deletes. Therefore, **cat** has a valid file name to read from, and we see the output of **ls -l**, taking one more step than usual to do so. Similarly, giving

>(*commands*) results in Bash naming a temporary pipe, which the commands inside the parenthesis read for input.

If you want to see whether two directories contain the same file names, run the single command:

```
cmp <(ls /dir1) <(ls /dir2)
```

The compare program **cmp** will see the names of two files which it will read and compare.

Command substitution also makes the **tee** command (used to view and save the output of a command) much more useful in that you can cause a single stream of input to be read by multiple readers without resorting to temporary files—bash does all the work for you. The command:

```
ls | tee >(grep foo | wc >foo.count) \
         >(grep bar | wc >bar.count) \
         | grep baz | wc >baz.count
```

counts the number of occurrences of **foo**, **bar** and **baz** in the output of **ls** and writes this information to three separate files. Command substitutions can even be nested:

```
cat <(cat <(cat <(ls -l)))
```

works as a very roundabout way to list the current directory.

As you can see, while the unnamed pipes allow simple commands to be strung together, named pipes, with a little help from bash, allow whole trees of pipes to be created. The possibilities are limited only by your imagination.

Andy Vaught is currently a PhD candidate in computational physics at Arizona State University and has been running Linux since 1.1. He enjoys flying with the Civil Air Patrol as well as skiing. He can be reached at andy@maxwell.la.asu.edu.

Archive Index Issue Table of Contents

Advanced search

Linux for Embedded Systems

Sandor Markon

Kenji Sasaki

Issue #41, September 1997

A company in Japan is using Linux for embedded systems in vertical transportation equipment such as elevators.

Fujitec is a global manufacturer and provider of a complete range of elevators, escalators and other vertical transportation equipment. Nearly all of our products use embedded computers in one form or other. To see the implications, a few features of the elevator business to note are:

- The lifetime of the product is very long, on the order of 20 years or more.
- Safety and availability of elevator service is a deciding factor for the users, especially for medium to high-rise buildings.
- Continuous service and maintenance during the full life cycle is vital; usually, it is even required by law.
- To meet the requirements of long term, disruption-free service and maintainability for embedded computers, we have recently started to investigate the possibility of using the freely available Linux OS and the GNU utilities.

The Target: Elevator Monitoring Equipment

We have in the past shipped several products using Linux. We will describe here only one of them—a monitoring display for group controllers (see Figure 1).

Figure 1. Controller Monitoring Display

Another application, a user-interface system for editing scrolling messages in elevator cars, is shown in Figure 2, but we do not have space to introduce it in detail. (Notice the Japanese characters—Linux works fine in Japan, too.)

Figure 2. Elevator User Interface

To put things into perspective, we need a few facts about elevators. If a building has more than 2 or 3 elevators, they are usually working in groups sharing the passenger traffic. A group of elevators is connected to a “Supervisory Group Controller”, which shares the service of each elevator among users while attempting to give each user the illusion that he/she has a personal elevator.

The modern elevator group controller is rather sophisticated; it has to run difficult, optimal-scheduling algorithms in real time. The latest Fujitec models use neural networks for on-line learning and optimization. Without proper tools, the support personnel who perform installation and testing tasks wouldn't understand what is happening with the elevator group.

For maintenance checks, upgrades and occasional troubleshooting, the GSP (Group Supervisory Panel) has a graphical display showing the status of the elevators. This monitoring display was the target of our first Linux application.

Figure 1 shows the graphic screen of the monitor display. Information for each elevator, such as the present floor, traveling direction, door status, registered calls, etc. is indicated by graphic symbols. This screen design has evolved over several years, and it is now familiar to our technical and maintenance personnel.

We built most of the original monitoring display system with custom-designed components:

- A custom graphic board with custom graphics software
- Control software common with our sequence-controller, CPU board
- Serial interface with our proprietary protocol to communicate with the group controller
- A multiscan VGA display (the only commercial part)

These components were chosen to give us the following advantages:

- Stand-alone operation in the elevator, machine-room environment
- Real-time response
- Long-term supply and maintenance
- Low cost

Even though we saw several problems with our custom-designed approach and wanted to move to an established, popular platform, until recently it was not feasible.

For the hardware, our first choice would have been the IBM- compatible PC, perhaps in the form of PC/104 cards. However, the dominant commercial OS (first DOS, then Windows), was unsuitable for many reasons. It is neither real-time capable nor reliable, and there is no chance that a particular release could have long-term support from its vendors.

Although there were many alternatives, such as the excellent Lynx OS, none of them satisfied all of our requirements, especially the long-term stability and availability criteria.

With the appearance of Linux, all this has changed.

Advantages and Problems of Using Linux for Embedded Systems

Over the past few years, several engineers at Fujitec became familiar with Linux. Although we started it as a hobby, gradually the wider applicability of Linux became clear to us.

Our motivation for embracing Linux for product development has several roots:

- The availability of the complete source tree of the system, without any restrictions on use, distribution or revisions, is vitally important to us. It can ensure that we will have some way of maintaining our systems at a site 5, 10, 15, ... years from now. If we wanted to guarantee the same with a commercial system, it could easily become a nightmare. We could imagine the vendor in 2010 (if still in business) just staring at us with mouth agape, when we ask them about bug fixes for their 1997 system. However, when we have the source, maintenance becomes “difficult” instead of “impossible”, just as with in-house, custom-made systems.
- An added bonus is the freedom from administration of a royalty-based, license system for some proprietary OS; not to mention the savings on the license fee.
- We have been using Unix for system development for many years. After some rather frustrating experiences with other systems on the PC platform, our developers were quite eager to get back to a Unix-like environment.
- Even among Unix systems, Linux feels unique: it is light, fast, runs on almost any platform, contains the latest version of all free software and is now so visible that even non-computer people are starting to talk about it. It is a good feeling to be back in the mainstream again.

However, not everything is rosy. Since Linux is essentially Unix, a few things had to be fixed before we could deploy it in an embedded controller.

- Ideally, we would like to use a ROM-based system, such as the one described in the article "Booting Linux from EPROM" by D. Bennett, *Linux Journal*, January, 1997. In our case, however, we needed X11, Tcl/Tk, libraries, fonts, etc., so we were forced to use a hard disk-based system. This means that we have to think about disk-buffer flushing, cleanup and recovery after messy shutdowns, etc.
- Linux is not a "hard" real-time system as it stands, although people are already working on real-time patches. (See "Introducing Real-Time Linux", M. Barabanov and V. Yodaiken, *Linux Journal*, February 1997.) For our current application, this was not directly a problem, but we would need guaranteed response time if we wanted Linux to control an elevator car.
- Installation must be made foolproof, especially for re-installation or upgrades. We have found that our customized procedure works, but it still needs to be improved.

The Linux Monitoring System

Figure 3. Monitoring System Block Diagram

Figure 3 shows the top-level block diagram of the monitoring system. We are using the Slackware 3.0 distribution with XFree86 for graphics. The application programs are compiled with the GNU C compiler (version 2.7.2). In some parts of a different application, we also use Tcl/Tk for our graphical user interface (GUI).

Many parts of the software, especially the X11 graphics, were ported without any problems from our existing Unix applications (under SunOS 4.1). A large part of our labor was spent on file system issues, in order to make the system installable and robust.

File System, Installation, Recovery

Our design is based on the UMSDOS file system, which resides over an MS-DOS-compatible host file system, e.g., Microsoft Windows. The decision to use UMSDOS took into account the reality of the proliferation of pre-installed Windows systems. Although Windows will not normally be used after Linux is up and running, we prefer not to remove it completely or to give Linux its own disk partitions.

With UMSDOS, the whole Linux system looks like a single MS- DOS directory tree from the Windows side. This fact gives us the option of doing installation and some restricted file maintenance from Windows, using CD-ROM or

floppies. However, we have found it faster and easier to boot Linux from floppy and stream the system to the hard disk through an Ethernet link. Network installation worked flawlessly both for desktop PCs, using a D-Link pocket adapter DL620, and for notebook PCs, using a Megahertz (US Robotics) PCMCIA network card.

The code we used to make the installation floppies, one each for the server (host) and the client (target) machines, is available by anonymous ftp at ftp://ftp.linuxjournal.com/pub/lj/listings/issue41/0133.tgz. Installation is almost completely automatic by starting the two networked PCs from floppy and answering a couple of questions about the X11 driver, display size etc. Because we have these floppies, the Linux file system with our applications can be "cloned" again and again from one Windows PC to the next.

Apart from installation, a major problem for our system is the crash recovery of the hard disk. Since we cannot guarantee orderly shutdowns, we have to deal with the cleanup after a "power failure"--like shutdown. We use the following techniques:

- Writing to the disk is restricted; most parts are made read only, and there is no swap file.
- At startup, the rc.local script runs the **umssync** program on the writable directories to repair possible inconsistencies.
- Possible garbage from a previous run is removed automatically.
- For emergency re-installation, a second clean copy of the full Linux system is maintained on the hard disk, and it is used to manually replace a damaged system.

In the future, we will use a RAM disk for the temporary files, and we hope to eventually fit the system into EPROMs.

Performance and Further Developments

We have installed the system both on notebook machines and on desktop computers. In case of the notebooks, the DSTN color display had a relatively narrow viewing angle, but it was usable.

The system was built into the group controller panel, and it has been tested under field conditions. We have also tested it at the research laboratory by connecting it to an elevator simulator that simulates a heavy load: 8 cars, 32 floors, all elevator cars moving constantly. All of this has proved to be a rather easy task for our system. There was no problem with either the serial interface, the graphics or the logging of the monitored data. Under the heaviest load conditions, we could log into the system and run other programs, even

recompile the application, without any effect on performance. Response was instantaneous, and it was evident that this platform can handle much more demanding applications.

We have tested the power failure and recovery capabilities and checked that the system can withstand almost any abuse.

As a future development, we are considering a system with a boot/root file system in ROM and a read-only, mounted hard disk for the large applications and libraries. Since we want to put a normal Linux file system on the hard disk, we need to develop kernel patches that let the system boot from a small root file system, and switch parts of it (/bin, /lib, etc.) during initialization.

Eventually, we would like to extend our tests to industrial grade PC systems and check the usability of Linux for continuous operation under strict environmental conditions. This would open the way for moving more critical monitoring, security and control tasks to the Linux/PC platform.

Conclusions

Linux is a viable and attractive platform for manufacturers who need a stable system that is “theirs to keep” for many years to come. Instead of developing in-house, custom-made systems at large cost or buying proprietary systems and leaving themselves at the mercy of the vendor, they can now choose Linux.

Linux, with the support of the Internet developer community and with guaranteed and affordable support from more and more commercial ventures, is seen by many as an ideal, software development platform. In our experience, it can also become an excellent deployment platform for computer-controlled hardware systems.

We are looking forward to the further expansion of our Linux business, and we hope the experience we gather will enable us to pay back a small part of our debt to the Linux community.

Sandor Markon graduated from the Technical University of Budapest and received a PhD in Electrical Engineering from Kyoto University. He has worked for Fujitec of Osaka, Japan since 1979 doing computer applications for elevator control. Present interests include neural networks, reinforcement learning, Internet programming with Java and industrial applications of free software. He can be reached via e-mail at markon@rd.fujitec.co.jp.

Kenji Sasaki graduated from Kyoto University, Faculty of Engineering. With Fujitec since 1976, he has been working mostly in control, systems

development and communications. Present interests include Java, Linux and the Internet. He can be reached via e-mail at kjsasaki@rd.fujitec.co.jp.

Archive Index Issue Table of Contents

Advanced search

New Products

Amy Kukuk

Issue #41, September 1997

OpenLinux Standard 1.1, TriTeal CDE for Linux, Diffpack and more.

OpenLinux Standard 1.1

Caldera Inc. announced the release of OpenLinux Standard 1.1. OpenLinux Stardard 1.1 is based on the new Linux kernel 2.0.29 and includes the Netscape FastTrack Server 2.0.1, Netscape Navigator 3.01 Gold, Sun Microsystem's Java Development Toolkit, Star Division's StarOffice 3.1 and Caldera's OpenDOS. OpenLinux is Caldera's platform for extending local area networks to the home, branch office, remote user and the Internet. All services on the local network can be extended around-the-corner or the world, across a high-speed connection by adding a frame relay or ISDN commodity card to Intel-based PCs. Suggested retail price for OpenLinux Standard is $399US.

Contact: Caldera Inc., 633 South 550 East, Provo, Utah, Phone: 801-229-1675, Fax: 801-229-1579, E-mail: info@caldera.com, URL: http://www.caldera.com/.

TriTeal CDE for Linux

Red Hat Software announced TriTeal CDE for Linux. Red Hat Software and TriTeal Corporation teamed up to bring you this common desktop environment. TriTeal provides users with a graphical interface to access both local and remote systems. Red Hat's TriTeal CDE for Linux is available in two versions. The Client Edition gives you everything you need to operate a complete licensed copy of the CDE desktop, incluidng the Motif 1.2.5 shared libraries. The Developer's Edition allows you to perform all functions of the Client Edition and includes a copy of OSF Motif version 1.2.5. CDE is an RPM-based product and installs easily on Red Hat and other RPM-based Linux systems. You can order on-line at http://www.redhat.com/ or call 1-888-REDHAT1.

Contact: Red Hat Software, Inc., 3203 Yorktown Avenue, Suite 123, Durham, Nc 27713, Phone: 800-546-7274, Fax: 919-572-6726, E-mail: info@redhat.com, URL: http://www.redhat.com/.

Diffpack

Numerical Objects announced the release of version 2.6.1 of Diffpack, an object-oriented C++ numerical library for building simulation software. The release covers the most common Unix platforms including Linux. Diffpack is a general purpose library for solving partial differential equations. It contains basic numerical entities, linear solvers, preconditioners and more. A stripped public domain source code version of Diffpack, restricted to academic use and evaluation, is available at http://www.nobjects.com/. The cost of a single user commercial license is $9000US.

Contact: Numerical Objects AS, Forskningsveien 1, P.O. Box 124 Blindern, N-0314 Oslo, Norway, Phone: 47-22-06-73-00, Fax: 47-22-06-73-51, E-mail: sea@nobjects.com, URL: http://www.nobjects.com/.

RIPmaster 2.0 for Linux

Advanced Systems Research announced the release of the first on-line graphical system to integrate with the Internet, while providing an on-line graphics protocol. RIPmaster 2.0 for Linux is a graphically based system using the extensive RIPscrip 3.0 protocol which enables the creation of graphics and multimedia systems. It runs under Linux on any supported hardware platform, including the DEC Alpha workstation, and supports a wide range of software. RIPmaster 2.0 is available for free. The RIPmaster demo system, RIPtel 3.0.2 and additional tools can be downloaded from http://www.telegrafix.com/asr/.

Contact: Advanced Systems Research, 2348 Appaloosa, West Linn. OR 97068, Phone: 503-650-5388, E-mail: amcnamee@cybernw.com, URL: http://www.telegrafix.com/asr/.

Ten X CD-ROM Server

Ten X Technology announced the 21 CD capacity TenXpert-4/21r. The TenXpert-4/21r offers 2GB of hard-disk cache with integrated server and five, 8x-speed, 4 CD microchangers and one 6x-speed read/4s-speed record CD recorder in a lockable tower. TenXpert has a plug-and-play design and performance is delivered by the large hard disk cache. Recorded CDs are immediately and automatically available to every user. The cost of the TenXpert-4/21r is $7245US.

Contact: Ten X Technology, Inc., 13091 Pond Springs Road, Suite B-200, Austin, TX, 78729, Phone: 800-922-9050, Fax: 512-918-9495, E-mail: info@tenx.com, URL: http://www.tenx.com/.

Clustor 1.1

Active Tools, Inc. announced the release of Clustor 1.1, a software tool for distributing and managing computationally intensive tasks. Clustor simplifies parametric executions—running the same application numerous times with different input parameters. Jobs can be distributed over a local area network or over the Internet. Clustor provides new features for load monitoring and resource sharing. Clustor 1.1 provides a graphical user interface for all phases of executing jobs on a network of computers. Clustor 1.1 for Linux is priced at $495US and $99US for additional computational nodes. Academic discount of 40% is available to academic institutions for non-commercial use.

Contact: Active Tools, 246 First Street, Suite 310, San Fransisco, CA, 94105, Phone: 415 882 7062, E-mail: info@activetools.com, URL: http://www.activetools.com/.

XMove 4.0 for Linux

Future Technologies and Siemens Austria ported the XMove 4.0 from RISC stations to Linux. XMove allows control of real-time processing using only a mouse click. XMove is software for designing, prototyping and testing the graphical user interface of a software system dealing with dynamically changing values. Xmove 4.0 for Linux, includes an Editor (Xmvdraw) and a complete library including the Motif Meter Object.

Contact: Future Technologies, Via Cairoli 1, Pordenone, Italy, Phone: 39-434-20-91-07, Fax: 39-434-20-95-10, E-mail: futuretg@tin.it, URL: http://www.vol.it/futuretec/.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

System Administration

Jan Rooijackers

Issue #41, September 1997

How to set up, use and maintain disk quotas for your Linux system.

Disk usage is always an issue, whether you are using Linux, DOS or any other Operating System. After discovering the disk is full the first thing to do is find out what files are taking up the most disk space and who owns those files. There are three different commands that you can use to obtain this information—**df**, **du** and **ls** (for more information, look in the man pages).

On using one of these commands, you find that one or more users have more disk space allocated than you do. One way for a System Administrator to avoid this kind of problem is to implement a disk quota for each user.

The Start

Before implementing the **quota** utility, you must have a kernel that supports it. **Quota** is supported by default in the Linux kernel since version 2.0. If you are not already running a 2.0.x kernel, you must install the **quota** package and create a new kernel that supports it.

Making Quota Available

To make **quota** available for a certain file system, you must edit the /etc/fstab file and add entries for **usrquota** and/or **grpquota**.

My fstab file is shown in Listing 1. The word **usrquota** is an option from the fstab that turns **quota** on for users on this device. You can also use **grpquota** to turn on **grpquota** for this device or use a combination of both.

Before you can use the **quota** package, the command **quotacheck** must be run to check the specified file system for any previously set quotas. If this is the first time you've used the command and no quotas are found, it creates a

quota.user or quota.group file or files in the root of the specified file system. Which files are created is dependent on which options are specified in the fstab file. The **quotacheck** command runs each time you boot the system.

The first time I ran **quotacheck** on my machine the output looked like this:

```
quotacheck -v /dev/hda3
Scanning /dev/hda3 [/home] done
Checked 50 directories and 331 files
Using quotafile /home/quota.user
```

Now that **quotacheck** has run, you can turn on **quota** for your system by using the command **quotaon**. This command has different options. The easiest one for first time use is:

```
quotaon -av
```

This command installs **quota** on all file systems marked read/write in the etc/fstab file and also displays a message showing which file systems have **quota** turned on. Here's another example:

```
newroom:~# quotaon -av
/dev/hda3: user quotas turned on
newroom:~#
```

To run **quotaon** each time you boot your machine, add the following line to the /etc/rc.d/rc.local file:

```
quotaon -avug
```

The opposite of **quotaon** is **quotaoff**, and it has the same options. This command turns **quota** off for a file system.

Giving Quotas

Now it is time to specify a quota for the users or groups. The easiest thing to do is to give everyone the same amount of disk space.

To get an indication of how much each user is currently using, use the command **repquota**. This command displays a summary of the disc usage and quotas for the specified file systems. For each user the current number of files and the amount of space (in kilobytes) is printed, along with any quotas created with **edquota** (see explanation of this command below). An example of this summary is shown in Listing 2.

The command used to set disk quotas is called **edquota**. This command brings up the **quota** editor which is used in the same way as the commands described above. The **-u** option is used to specify a user quota, and the **-g** option is used to specify a group quota. When you use **edquota** with one of the options, a temporary file is created containing an ASCII representation of the current disk

quotas for that user or group, and the editor is invoked for this file. You can use the editor to modify or add new quotas and so on. Upon exiting the editor, **edquota** reads the temporary file and modifies the binary **quota** files to reflect the changes. When in the editor, you should only edit numbers that follow an **=** sign. For each file system using **quota**, two lines are put in the temporary file:

```
Quotas for user dsnjaro:
/dev/hda3: blocks in use: 49, limits (soft = 0, hard = 0)
        inodes in use: 30, limits (soft = 0, hard = 0)
```

The first line contains the number of blocks in use and how many blocks a user or group can allocate. The second line contains the number of inodes in use and how many can be allocated. The **soft** parameter specifies a “soft limit”—people or groups can exceed this limit for a certain period of time (set by the **-t** option). The **hard** parameter specifies a “hard limit”—the absolute maximum amount of space a user or group can have.

If you don't wish to set a quota for a particular user or group, assign the value 0 to both **soft** and **hard**. This is a better documented solution than leaving this user or group out of the user.quota or group.quota file.

To change the hard and soft limits, use the **edquota** command with the **-t** option set. Using the editor, you can specify these time limits in either days, hours, minutes or seconds. If you set the hard limit equal to the soft limit, users or groups are not allowed to have more than this value.

To give everyone on your system the same quota, use the **-p** option to define a prototype user. To give everyone the same quota as this prototype user, give the command:

```
edquota -p <uid of the prototype> *
```

All commands described in this article are only for use by the system administrator (almost every command has to read all directories and their files) for security reasons.

How Users Can Check Their Quotas

Every user or group can check their disk quota with the command **quota**. This command produces a report that contains information for all file systems listed in the /etc/fstab. Give the command **quota -u** (for user quota) or **quota -g** (for group quota) or a combination to obtain this information. If no quotas are set, the command **quota -u** results in the following output:

```
Disk quotas for dsnjaro (uid 503):none
```

If quotas have been set, the output looks like:

```
Disk quotas for user dsnjaro (uid 503):
File system blocks  quota  limit  grace  files  quota  limit grace
/dev/hda3   49   100   110       30    0    0
```

Only the system administrator can use either of the commands:

```
quota -u <
quota <-g <group-id>
```

Quota is working very well for me in my work, where there are approximately 300 users on our system.

Jan Rooijackers works at Ericsson Data Netherlands as an Information Systems Engineer. His first contact with Unix was in 1991 and with Linux in 1994. He likes to spend time with his family and his PCs. He can be reached via e-mail at Jan.Rooijackers@dsn.ericsson.se.

Archive Index Issue Table of Contents

Advanced search

The sysctl Interface

Alessandro Rubini

Issue #41, September 1997

A look at the sysctl system call that gives you the ability to fine tune kernel parameters.

The **sysctl** system call is an interesting feature of the Linux kernel; it is quite unique in the Unix world. The system call exports the ability to fine-tune kernel parameters and is tightly bound to the /proc file system, a simpler, file-based interface that can be used to perform the same tasks available by means of the system call. **sysctl** appeared in kernel 1.3.57 and has been fully supported ever since. This article explains how to use sysctl with any kernel between 2.0.0 and 2.1.35.

When running Unix kernels, system administrators often need to fine-tune some low-level features according to their specific needs. Usually, system tailoring requires you rebuilding the kernel image and rebooting the computer. These tasks are lengthy ones which require good skills and a little luck to be successfully completed. Linux developers diverged from this approach and chose to implement variable parameters in place of hardwired constants; run-time configuration can be performed by using the **sysctl** system call or more easily by exploiting the /proc file system. The internals of **sysctl** are designed not only to read and modify configuration parameters, but also to support a dynamic set of such variables. In other words, the module writer can insert new entries in the sysctl tree and allow run-time configuration of driver features.

The /proc Interface to System Control

Most Linux users are familiar with the /proc file system. In short, the file system can be considered a gateway to kernel internals: its files are entry points to certain kernel information. Such information is usually exchanged in textual form to ease interactive use, although the exchange can involve binary data when required. The typical example of a binary /proc file is /proc/kcore, a core file that represents the current kernel. Thus, you can execute the command:

```
gdb /usr/src/linux/vmlinux /proc/kcore
```

and peek into your running kernel. Naturally, **gdb** on /proc/kcore gives much better results if vmlinux has been compiled using the **-g** compiler option.

Most of the /proc files are read-only: writing to them has no effect. This applies, for instance, to /proc/interrupts, /proc/ioports, /proc/net/route and all the other information nodes. The directory /proc/sys, on the other hand, behaves differently; it is the root of a file tree related to system control. Each subdirectory in /proc/sys deals with a kernel subsystem like net/ and vm/, while the kernel/ subdirectory is special as it includes kernel-wide parameters, like the file kernel/hostname.

Each sysctl file includes numeric or string values—sometimes a single value, sometimes an array of them. For example, if you go to the /proc/sys directory and give the command:

```
grep . kernel/*
```

kernel 2.1.32 returns data similar to the following:

```
kernel/ctrl-alt-del:0
kernel/domainname:systemy.it
kernel/file-max:1024
kernel/file-nr:128
kernel/hostname:morgana
kernel/inode-max:3072
kernel/inode-nr:384     263
kernel/osrelease:2.1.32
kernel/ostype:Linux
kernel/panic:0nn
kernel/printk:6   4    1   7
kernel/securelevel:0
kernel/version:#9 Mon Apr 7 23:08:18 MET DST 1997
```

It's worth stressing that reading /proc items with **less** doesn't work, because they appear as zero-length files to the **stat** system call, and **less** checks the attributes of the file before reading it. The inaccuracy of **stat** is a feature of /proc, rather than a bug. It's a saving in human resources (in writing code), and kernel size (in carrying the code around). **stat** information is completely irrelevant for most files, as **cat**, **grep** and all the other tools work fine. If you really need to use **less** to look at the contents of a /proc file, you can resort to:

```
cat
```

If you want to change system parameters, all you need to do is write the new values to the correct file in /proc/sys. If the file contains an array of values, they will be overwritten in order. Let's look at the kernel/printk file as an example. **printk** was first introduced in kernel version 2.1.32. The four numbers in /proc/sys/kernel/printk control the “verbosity” level of the **printk** kernel function. The first number in the array is **console_loglevel**: kernel messages with priority less than or equal to the specified value will be printed to the system console (i.e.,

the active virtual console, unless you've changed it). This parameter doesn't affect the operation of **klogd**, which receives all the messages in any case. The following commands show how to change the log level:

```
# cat kernel/printk
6       4       1       7
# echo 8 > kernel/printk
# cat kernel/printk
8       4       1       7
```

A level of 8 corresponds to debug messages, which are not printed on the console by default. The example session shown above changes the default behaviour so that every message, including the debug ones, are printed.

Similarly, you can change the host name by writing the new value to /proc/kernel/hostname—a useful feature if the **hostname** command is not available.

Using the System Call

Even though the /proc file system is a great resource, it is not always available in the kernel. Since it's not vital to system operation, there are times when you choose to leave it out of the kernel image or simply don't mount it. For example, when building an embedded system, saving 40 to 50KB can be advantageous. Also, if you are concerned about security, you may decide to hide system information by leaving /proc unmounted.

The system call interface to kernel tuning, namely sysctl, is an alternative way to peek into configurable parameters and modify them. One advantage of sysctl is that it's faster, as no fork/exec is involved (i.e., no external programs are spawned) nor is any directory lookup. However, unless you run an ancient platform, the performance savings are irrelevant.

To use the system call in a C program, the header file sys/sysctl.h must be included; it declares the sysctl function as:

```
int sysctl (int *name, int nlen, void *oldval,
  size_t *oldlenp, void *newval, size_t newlen);
```

If your standard library is not up to date, the **sysctl** function will neither be prototyped in the headers nor defined in the library. I don't know exactly when the library function was first introduced, but I do know libc-5.0 does not have it, while libc-5.3 does. If you have an old library you must invoke the system call directly, using code such as:

```
#include <linux/unistd.h>
#include <linux/sysctl.h>
/* now "_sysctl(struct __sysctl_args *args)"
   can be called */
_syscall1(int, _sysctl, struct __sysctl_args *,
        args);
```

The system call gets a single argument instead of six of them, and the mismatch in the prototypes is solved by prepending an underscore to the name of the system call. Therefore, the system call is **_sysctl** and gets one argument, while the library function is **sysctl** and gets six arguments. The sample code introduced in this article uses the library function.

The six arguments of the **sysctl** library function have the following meaning:

1. **name** points to an array of integers: each of the integer values identifies a sysctl item, either a directory or a leaf node file. The symbolic names for such values are defined in the file linux/sysctl.h.
2. **nlen** states how many integer numbers are listed in the array **name**. To reach a particular entry you need to specify the path through the subdirectories, so you need to specify the length of this path.
3. **oldval** is a pointer to a data buffer where the old value of the sysctl item must be stored. If it is **NULL**, the system call won't return values to user space.
4. **oldlenp** points to an integer number stating the length of the **oldval** buffer. The system call changes the value to reflect how much data has been written, which can be less than the buffer length.
5. **newval** points to a data buffer hosting replacement data. The kernel will read this buffer to change the sysctl entry being acted upon. If it is **NULL**, the kernel value is not changed.
6. **newlen** is the length of **newval**. The kernel will read no more than **newlen** bytes from **newval**.

Now, let's write some C code to access the four parameters contained in /proc/sys/kernel/printk. The numeric name of the file is KERN_PRINTK, within the directory CTL_KERN/ (both symbols are defined in linux/sysctl.h). The code shown in Listing 1, pkparms.c, is the complete program to access these values.

Changing sysctl values is similar to reading them—just use **newval** and **newlen**. A program similar to pkparms.c can be used to change the console log level, the first number in kernel/printk. The program is called setlevel.c, and the code at its core looks like:

```
int newval[1];
int newlen = sizeof(newval);
/* assign newval[0] */
error = sysctl (name, namelen, NULL /* oldval */,
         0 /* len */, newval, newlen);
```

The program overwrites only the first **sizeof(int)** bytes of the kernel entry, which is exactly what we want.

Please remember that the printk parameters are not exported to **sysctl** in version 2.0 of the kernel. The programs won't compile under 2.0 due to the missing KERN_PRINTK symbol; also, if you compile either of them against later versions and then run under 2.0, you'll get an error when invoking sysctl.

The source files for pkparms.c, setlevel.c and hname.c (which will be introduced in a while) are in the 2365.tgz1 file.

A simple run of the two programs introduced above looks like the following:

```
# ./pkparms
len is 16 bytes
6       4       1       7
# cat /proc/sys/kernel/printk
6       4       1       7
# ./setlevel 8
# ./pkparms
len is 16 bytes
8       4       1       7
```

If you run kernel 2.0, don't despair—the files acting on kernel/printk are just samples, and the same code can be used to access any sysctl item available in 2.0 kernels with minimal modifications.

On the same ftp site you'll also find hname.c, a bare-bones **hostname** command based on sysctl. The source works with the 2.0 kernels and demonstrates how to invoke the system call with no library support, since my Linux-2.0 runs on a libc-5.0-based PC.

A Quick Look at Some sysctl Entries

Although low-level, the tunable parameters of the kernel are very interesting to tweak and can help optimize system performance for the different environments where Linux is used.

The following list is an overview of some relevant /kernel and /vm files in /proc/sys. (This information applies to all kernels from 2.0 through 2.1.35.)

- **kernel/panic** - The integer value is the number of seconds the system will wait before automatic reboot in case of system panic. A value of 0 means “disabled”. Automatic reboot is an interesting feature to turn on for unattended systems. The command-line option **panic=*value*** can be used to set this parameter at boot time.
- **kernel/file-max** - The maximum number of open files in the system. **file-nr**, on the other hand, is the per-process maximum and can't be modified, because it is constrained by the hardware page size. Similar entries exist for the inodes: a system-wide entry and an immutable per-process one.

Servers with many processes and many open files might benefit by increasing the value of these two entries.

- **kernel/securelevel** - This is a hook for security features in the system. The securelevel file is currently read-only even for root, so it can only be changed by program code (e.g., modules). Only the EXT2 file system uses securelevel—it refuses to change file flags (like **immutable** and **append-only**) if securelevel is greater than 0. This means that a kernel, precompiled with a non-zero securelevel and no support for modules, can be used to protect precious files from corruption in case of network intrusions. But stay tuned for new features of securelevel.
- **vm/freepages** - Contains three numbers, all counts of free pages. The first number is the minimum free space in the system. Free pages are needed to fulfill atomic allocation requests, like incoming network packets. The second number is the level at which to start heavy swapping, and the third is the level to start light swapping. A network server with high bandwidth benefits from higher numbers in order to avoid dropping packets due to free memory shortage. By default, one percent of the memory is kept free.
- **vm/bdflush** - The numbers in this file can fine-tune the behaviour of the buffer cache. They are documented in fs/buffer.c.
- **vm/kswapd** - This file exists in all of the 2.0.x kernels, but has been removed in 2.1.33 as not useful. It can safely be ignored.
- **vm/swapctl** - This big file encloses all the parameters used in fine-tuning the swapping algorithms. The fields are listed in include/linux/swapctl.h and are used in mm/swap.c.

The Programming Interface: Plugging New Features

Module writers can easily add their own tunable features to /proc/sys by using the programming interface to extend the control tree. The kernel exports to modules the following two functions:

```
struct ctl_table_header *
        register_sysctl_table(ctl_table * table,
        int insert_at_head);
void unregister_sysctl_table(
        struct ctl_table_header * table);
```

The former function is used to register a “table” of entries and returns a token, which is used by the latter function to detach (unregister) your table. The argument **insert_at_head** tells whether the new table must be inserted before or after the other ones, and you can easily ignore the issue and specify 0, which means “not at head”.

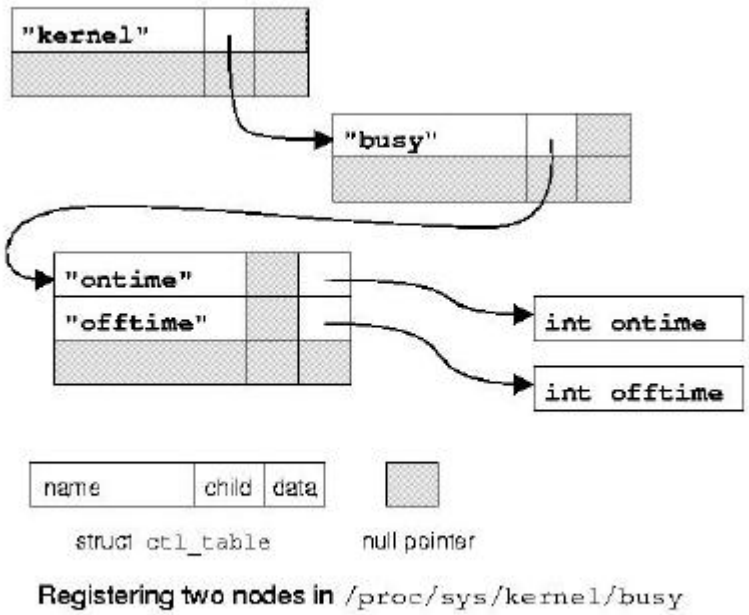
What is the **ctl_table** type? It is a structure made up of the following fields:

- **int ctl_name** - This is a numeric ID, unique within each table.
- **const char *procname** - If the entry must be visible through /proc, this is the corresponding name.
- **void *data** - The pointer to data. For example, it will point to an integer value for integer items.
- **int maxlen** - The size of the data pointed to by the previous field; for example, **sizeof(int)**.
- **mode_t mode** - The mode of the file. Directories should have the executable bit turned on (e.g., **0555** octal).
- **ctl_table *child** - For directories, the child table. For leaf nodes, **NULL**.
- **proc_handler *proc_handler** - The handler is in charge of performing any read/write spawned by /proc files. If the item has no **procname**, this field is not used.
- **ctl_handler *strategy** - This handler reads/writes data when the system call is used.
- **struct proc_dir_entry *de** - Used internally.
- **void *extra1, *extra2** - These fields have been introduced in version 1.3.69 and are used to specify extra information for specific handlers. The kernel has an handler for integer vectors, for example, that uses the extra fields to be notified about the allowable minimum and maximum allowed values for each number in the array.

Well, the previous list may have scared most readers. Therefore, I won't show the prototypes for the handling functions and will instead switch directly to some sample code. Writing code is much easier than understanding it, because you can start by copying lines from existing files. The resulting code will fall under the GPL—of course, I don't see that as a disadvantage.

Let's write a module with two integer parameters, called **ontime** and **offtime**. The module will busy-loop for a few timer ticks and sleep for a few more; the parameters control the duration of each state. Yes, this *is* silly, but it is the simplest hardware-independent example I could imagine.

The parameters will be put in /proc/sys/kernel/busy, a new directory. To this end, we need to register a tree like the one shown in Figure 1. The /kernel directory won't be created by **register_sysctl_table**, because it already exists. Also, it won't be deleted at **unregister** time, because it still has active child files; thus, by specifying the whole tree of directories you can add files to every directory within /proc/sys.



Registering two nodes in /proc/sys/kernel/busy

Listing 2 is the interesting part of **busy.c**, which does all the work related to sysctl. The trick here is leaving all the hard work to **proc_dointvec** and **sysctl_intvec**. These handlers are exported only by version 2.1.8 and later of the kernel, so you need to copy them into your module (or implement something similar) when compiling for older kernels.

I won't show the code related to busy looping here, because it is completely out of the scope of this article. Once you have downloaded the source from the FTP site1, it can be compiled on your own system. It works with both version 2.0 and 2.1 on the Intel, Alpha and SPARC platforms.

Probing Further

Despite the usefulness of **sysctl**, it's hard to find documentation. This is not a concern for system programmers, who are accustomed to peeking at the source code to extract information. The main entry points to the sysctl internals are kernel/sysctl.c and net/sysctl_net.c. Most items in the sysctl tables act on solely on strings or arrays of integers. So to search through the whole source tree for an item, you will end up using the **data** field as the argument to **grep**. I see no shortcut to this method.

As an example, let's trace the meaning of **ip_log_martians** in /proc/sys/net/ipv4. You'll first find that sysctl_net.c refers to **ipv4_table**, which in turn is exported by sysctl_net_ipv4.c. This file in turn includes the following entry in its table:

```
{NET_IPV4_LOG_MARTIANS, "ip_log_martians",
&ipv4_config.log_martians, sizeof(int), 0644,
NULL, &proc_dointvec},
```

Understanding the role of our control file, therefore, reduces to looking for the field **ipv4config.log_martians** throughout the sources. Some grepping will show that the field is used to control verbose reporting (via **printk**) of erroneous packets received by this host.

Unfortunately, many system administrators are not programmers and need other sources of information. For their benefit, kernel developers sometimes write a little documentation as a break from writing code, and this documentation is distributed with the kernel source. The bad news is that, sysctl is quite recent in design, and such extra documentation is almost nonexistent.

The file Documentation/networking/Configurable is a short introduction to **sysctl** (much shorter than this article) and points to net/TUNABLE, which in turn is a huge list of configurable parameters in the network subtree. Unfortunately the description of each item is quite technical, so that people who don't know the details of networking can't proficiently tune network parameters. As I'm writing, this file is the only source of information about system control, if you don't count C source files.

Alessandro Rubini reads e-mail as rubini@linux.it and enjoys breeding oaks and playing with kernel code. He is currently looking for a job in either field.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #41, September 1997

Our experts answer your technical questions.

The .plan File

What are the .plan and .project files read by the **finger** command and what should they contain? —Chris MasonSlackware 2.0.29

In the good old days before the Web, there were no home pages. Instead, if I wanted to find out more about someone, I would “finger” them. The **finger** command asks the server to display information about a user, including the contents of the user's .plan and .project files. You can put any information you wish in these files: your name, e-mail address, fax and phone numbers or favorite sayings.

Note that many system administrators consider the **finger** command to be a potential security risk and have turned it off, so don't be surprised if you “finger” someone and receive a message along the lines of “access denied”. Also, many implementations of **finger** read only the .plan file. —Vince Waldon vwaldon@skynet.uah.ualberta.ca

Linux and Pentiums

I have a PC with an Intel Pentium 150. Will Linux run on it? I've heard it runs on a 386 or a 486 but has trouble with certain IBMs—I'm not sure which ones. —Noah Roberts

You should have no problem running Linux on the machine you describe. Early versions of the Linux kernel were unable to support true IBM machines that used the microchannel architecture or MCA (the PS\2 line). That's probably the IBM computer referred to. —Vince Waldon vwaldon@skynet.uah.ualberta.ca

Spinning Hard Drive

My hard drive spins up and spins down constantly. There is a kernel patch called **no_idle** on Sunsite [http://sunsite.unc.edu/] to fix this problem, but when I attempt to apply the patch I get a reject file. It seems that the Makefile for the disk drivers has now been created, and as a result, the patch does not apply correctly. I am running 2.0.0. I would like to know if there is something else I could do to stop the spinning. I would appreciate any help. —John BarnitzSlackware 3.1

Most likely you need to find the hdparm package and use it to set the spin down times. I know it can do this for IDE drives. If it's not part of your distribution, you can find it on Sunsite. —Donnie Barnes, Red Hat Software redhat@redhat.com

SCSI Drivers

Are there any drivers that provide SCSI support on the motherboard? —Ryan Red Hat 4.1

That depends on the type of SCSI you wish to use. You can check the hardware compatibility lists at http://www.redhat.com/. —Donnie Barnes, Red Hat Software redhat@redhat.com

Non-English Keyboard Characters

After installing Linux, I noticed I am missing an option available in MS Windows: the US-International keyboard layout. This layout lets anyone with a US keyboard type the special punctuation needed for foreign languages. I live in Puerto Rico, and most, if not all, keyboards sold here are US versions. Since I write mostly in Spanish, I am interested in learning how to make a keymap that emulates Windows' US-International layout. Is there any information about the subject or any already-made keymap file that fits the job? —Carlos M. Fernandez Red Hat 4.1

There may indeed be a keyboard mapping that fits your keyboard. If not, you will have to take one that is close to your desired arrangement and modify it.

You should obtain the **kbd** package from ftp://sunsite.unc.edu/pub/Linux/system/keyboards/kbd-0.98.tar.gz. It contains tools, documentation and examples that will assist you in your remapping project. It also contains a file called kbd.FAQ, which contains answers to frequently asked questions about the operation of the keyboard under Linux. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Configuring su for Security

I have a problem with hackers and one security hole is the command **su**. I have several users on my system. While I don't want to eliminate the capability of these users to change to other IDs, I do want to eliminate the capability to use **su** to change to root for all except one or two users. Is this possible? —Are Tysl Slackware 3.1

You may be missing a handy program called **sudo**, which you can obtain from your nearest Sunsite mirror. This program allows you to configure **su** actions for each user based on who the user is and what you wish him to be able to access.

If that does not meet your goal, why not fall back to the standard Unix security functions? Create a new group called su. Change the group on /bin/su from bin to su. The permissions are most likely 4755 (-rwsr-xr-x), which means anybody can execute it and the program will execute as root.bin.

You can then change the permissions of /bin/su. Try changing them to 4750 (-rwsr-x---), which allows root or any user in the su group to execute it. Then you can put those users you wish to have **su** privileges in the su group. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Man Page Display

How do I use **man**? For example, when I enter:

```
man ls
```

I get a blank screen with a weird message at the bottom of the screen—something like 1/1. Whatever I enter, it beeps at me. —Josh Gray Slackware 3.2

Check whether there are any files in the /usr/man/manx directory (where x is a number, usually from 1 to 8). You should find several different files with names like gpm.1. Each of these files is a man page. Whenever you use the **man** command, you get a processed version of the file corresponding to the command specified (for the **ls** command, it is the ls.x file). For this file to be processed, the **groff** utility must be installed. **groff** is usually found in the /usr/bin directory. —Mario de Mello Bittencourt Neto, WebSlave mneto@buriti.com.br

Setting Up Swap Space

When I installed Linux, I didn't set up a swap space. I have since created a swap file but I have to enter:

```
swapon /dev/hda5
```

every time I boot, and I can do it only as root. Can I make this simpler? —Josh Gray Slackware 3.2

Slackware puts entries to automatically mount swap partitions (if they exist) in your rc script files. All you need to do is tell those files that your swap partition exists and is available for use. To do that, put a line in the /etc/fstab file like the following:

```
/dev/hda5    swap    swap    defaults    1   1
```

This tells the system to set up a swap space from /dev/hda5 with the default settings for a swap partition. This entry is normally created by the setup scripts when you install Slackware, and is the missing item that prevents your swap area from being initialized with each boot. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Archive Index Issue Table of Contents

Advanced search