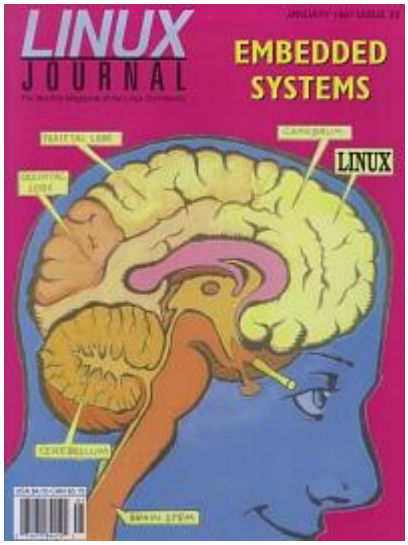
Linux Journal Issue #33/January 1997



Features

Let Linux Speak *by David Sugar*
How an ad for a speech synthesizer led to the the development of a speech server under Linux.

Booting Linux from EPROM *by Dave Bennett*
A quick look at making Linux bootable from EPROM on a 486single board computer.

Using Linux with Programmable Logic Controllers *by J.P.G. Quintana*
Combining programmable logic controllers with linux can be acost-effective and robust method for providing specializedcontrol systems.

News & Articles

Disk Maintenance under Linux (Disk Recovery) *by David A Bandel*
Satellite Tracking with Linux *by Kenneth E Harker*
Free SCO OpenServer Has Its Place *by Evan Leibovitch*
Caldera's Bryan Sparks *by Phil Hughes*

Reviews

Product Review Netactive SynergieServer Pro *by Jonathan Gross*

WWWsmith

Java and Client-Server *by Joe Novosel*
At the Forge CGI Programming *by Reuven Lerner*

Columns

Take Command unzip *by Greg Roelofs*
New Products
Linux Gazette Two Cent Tips *by Marjorie Richardson*

Archive Index

Advanced search

Let Linux Speak

David Sugar

Issue #33, January 1997

Set up a speech server on your PC Linux machine with the Computalker.

“User root is now on-line”. Words to be dreaded when one is away from the terminal, and not logging in otherwise. But how does one know what is going on with one's machine when not in front of it? If only the machine could tell you. In this article I discuss a tool which enables your machine to do just that.

It all started a year back, when, thumbing through one of those odd electronic magazines, I came across an ad for a little speech synthesizer. This device was essentially a low cost serial-based text-to-speech synthesizer using the SPO256-AL2 chip. I believe this was the same chip used in the original Mattel “Speak & Spell” toy.

After a couple of months, I thought about it again and decided I just had to have it. Certainly, the price was right (about US $50.00), and serial ports grew on my main Linux machine like branches on a tree. So I ordered one. After a few weeks, I called and was told my order had just been hand-made and would be out in a few days. It is a delight to find hand-made electronics in these modern times—almost like the days when furniture manufacturing involved real craftsmanship.

In any case, the unit arrived as promised, complete with schematics, a disk filled with DOS programs and a thin manual. The disk I have yet to look at; after all, this was for use with a Linux machine. The board slid into a PC slot easily enough. The card uses the PC slot for power only. An RS-232 connector in the back connects to a serial port. A separate stand-alone power unit and case is available for $29.00 more. But having another power pack to plug in was enough to keep me awake at night. A slot I could afford; though I now foresee the time when I will fill up all eight slots in the machine.

The board has its own built-in speaker and an RCA jack. The RCA jack I quickly adapted to feed the background music (BGM) source on my PBX at home. (Okay, so it's really a Panasonic digital hybrid key system, to be technical, although it has ambitions.) I connected the serial port and got a brief noise as DTR was raised. I shortly learned this was supposed to say “Okay”, but the impedance-matching on the RCA jack was poor.

Next, I changed the stty settings on the port to match the speed I had selected for the device via dip switches, and, with high expectations, I tried a simple test:

```
echo "Hello, my name is Rochester" >/dev/ttyS2
```

The monotone response I received back sounded a little like “Hewlo, my name is Rokheestar” and reminded me of my last visit to Atlanta, where they use a deliberately harsh-sounding cybernetic voice on the inter-terminal shuttle trains. Hmmm, maybe it is time to look at the manual, and maybe even that disk...

Several limitations and problems became immediately obvious. The first was the text-to-speech algorithms handled words only. Numbers are simply spoken as a series of digits. Hence 91 becomes “nine one”, instead of “ninety-one”. This can be solved by some simple look-up tables and text substitution.

Second, while technically the device acts as a text-to-phonetic speech device, no special means, such as control or escape sequences, allow direct access to the phonetic elements and sounds the device can produce; the text-to-speech code hides them. This second limitation can be resolved by using alternate spellings, though not necessarily phonetic spellings, that saturate the internal algorithm toward different phonetic choices. A little experimentation was required to get a good idea of how the device actually translated text to speech.

Since extensive table substitution was now needed, I considered the next logical step; to develop a driver as a front end for the device. Ideally, any driver should be able to read straight text the way a person normally would. First, numbers should be pronounced as numbers and not as digits. Similarly, many common numeric constructs used in normal text—such as currency amounts, standard formatted date and time fields, percentages, telephone numbers, etc. —have pronunciation rules I wished to encapsulate and emulate properly. The Internet has its own idioms, like x@y.z, which should be pronounced as “x at y dot z”. I decided to cover all of these, as well as in-line text substitution for correct word pronunciation.

In the end, I decided on a server sitting on a TCP socket. The server would accept a connection from the user application on a known port and pronounce any text received according to a reasonable set of rules (as stated above). I

added an escape mode to allow for spelling words out and single-digit announcement modes. I could establish a simple telnet session with the server, then test the device by typing text.

The TCP server offered another advantage. Only one application can be serviced by the device at a time—otherwise speech would be garbled together from multiple sources. The use of a TCP session assures that only one connection would be accepted by the server and kept active until closed by the client. Other client applications can block as backlog while waiting for the current application to finish talking. The simplicity offered by backlogging, over the use of lock files was the reason I chose to use a full server instead of a task initiated by inetd.

With the server in place, it was only a matter of time before speech synthesis would pervade other system services. The first use I made of the server was to monitor my BBS system. By connecting it to the user login quota manager, I could have the device announce as users logged in and out. Similarly, the traditional sysop page can be carried over this device.

Eventually I tied the SPO server into my implementation of the **wall** command and then created other utilities to provide verbal monitoring of my Internet server. Verbal monitoring would watch for and announce new e-mail for me, as well as basic system stats such as uptime and disk usage every hour. As all this speech can be annoying at night, I added a simple muting schedule to the server. Most curious and entertaining is my replacement for shutdown, called simply “down”.

For system monitoring, the speech device has proven to be quite a useful tool —not a nuisance. The server was developed for the ability to read written text and properly pronounce common usages and conventions, and while I use this capability minimally, others might have more occasion for it. The pronunciation dictionary can be expanded as needed to cover a wider range of words as they are identified in everyday use.

One use for the device which was suggested to me is as a screen reader for visually-impaired computer users. Another application I am looking at is in parking incoming phone calls and paging or announcing calls through the telephone system. I have often wished the board included a DTMF tone generator and a SLICK, so I may look at modifying the schematics provided.

The SPO-256-AL2 text-to-speech board described here may be purchased through B.G. Micro, P.O. Box 280298, Dallas, TX 75228 (214) 271-5546. The Computalker lists for around $50.00 (U.S.) as a PC card or $80.00 (U.S.) stand-

alone with a power adapter. Chips are available separately, and I believe the Computalker may be purchased in kit form.

While the SPO is serial-based and can be used on almost any machine or OS, I originally obtained it for use on my main server, which runs Linux. For this reason, the speech server was developed and tested under Linux. The server was originally developed using libraries and part of the code base of my BBS package, so these are included as part of the published source. I am working on a more portable public source implementation that should be more easily and widely compatible to non-Linux systems as well. I must go now, as I am being paged...

Code for the Synthesizer

David Sugar Best known for WorldVU, a public BBS system for Linux, he is currently employed as director of software engineering for Fortran Corp. and uses Linux for commercial telephony development. He maintains his own Internet server under Linux.

Archive Index Issue Table of Contents

Advanced search

Booting Linux from EPROM

Dave Bennett

Issue #33, January 1997

This is a quick look at making Linux bootable from EPROM on a 486 single board computer.

This article describes one way to run Linux in an embedded system with no hard disk. The application described is an Operator Interface in a monitor and display system developed by Boeing Flight Test. The airborne environment requires something fairly rugged which can withstand common power interruptions. To meet these requirements we decided to build the operator interface without a hard disk.

Overview

The basic concept includes booting from a solid state disk (SSD) in **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory (EPROM), copying a root file system from EPROM to a RAM disk, loading the operator interface software from a host and executing it. This article focuses on the details of how the system works, and on development techniques used.

The hardware selected is a VME-based Single Board Computer (SBC) 80486 with 16M of RAM, a PC104 SSD cable of holding a 4Meg EPROM, and some other PC104 boards. This SBC has built in BIOS support for using the SSD. The system uses a programmable keyboard and a standard VGA display.

System Operation

For booting, two options were considered:

- booting DOS, then running the loadlin program (to load Linux) from autoexec.bat
- installing LILO and booting Linux directly

The advantage of the second option would be a slightly shorter boot time. However, we implemented the first option, because we wanted to use a programmable keyboard—the software for programming the keyboard runs under DOS.

A bit of kernel-hacking was needed to make the system work. The ramdisk.c code was changed to load from any block device, not just a floppy (see Listing 1, ramdisk.c). Also, a new block driver was written to read from the EPROM device (see Listing 2, epromdsk.c).

When deciding how to implement the EPROM device driver, the first idea was to create an image of a disk in the EPROM. This would provide a RAM disk of the same size as the EPROM, 3.5MB in this case (the DOS portion of the SSD takes 1/2 MB). Instead, to allow a larger RAM disk, a compressed disk image is used. The compression used is simple—any sectors which are identical are only stored once. The primary advantage this gives is blank areas of the disk image don't need to take up EPROM space. Listing 1 shows the SSD disk compression used.

EPROM Disk Compression

In order to automatically run the operator interface application, a program was written to replace getty. This program (dboot.c) will run login for a given user, and set the stdin, stdout and stderr to the specified virtual console.

The boot sequence is:

- Power up and run memory tests
- load DOS which executes AUTOEXEC.BAT
    - run the keyboard programming application
    - run loadlin—this reads a linux kernel from the DOS disk & executes it
- the linux kernel takes over
- load the RAM disk from the EPROM disk
- switch the root file system to the RAM disk
- init will read inittab which executes dboot instead of getty
- the operator interface application is started

Development

After the fun part—figuring how to make an EPROM driver and how to boot the system—the more mundane task of putting together the EPROM disk contents

had to be done. This was done using a development disk which was partitioned as follows:

- /dev/hda1 - 80MB “full” Linux system
- /dev/hda2 - 6MB EPROM development
- /dev/hda3 - 20MB DOS partition
- LILO was used to allow booting of either Linux or DOS.

Programming EPROMs is a time-consuming task and to be avoided as much as possible. As a result, most of the development is done using the disk.

The first phase of disk image development was identifying the required and the desired items. The first step was to come up with a minimal system and then add the items required for the operator interface. Not being a Unix expert, coming up with the minimal system ended up being something of a trial and error process. I started with what I thought was needed, then tried running it. When an error occurred because of a missing program or library, that file was added. This process went on until the system ran happily.

The bulk of this was done by copying files from the “full” Linux partition to the 6MB partition, booting DOS and using the loadlin line:

```
loadlin zimage root=/dev/hda2 ro
```

Once the system was fairly stable, the 6MB partition was loaded into the RAM disk. This is very similar to how the RAM disk is loaded from EPROM, but development went faster since EPROMs weren't being programmed. To test the system without programming EPROMs, the system booted DOS and ran loadlin with the line:

```
loadlin zimage root=/dev/hda2 ramdisk=6144 ro
```

Because of the modification to ramdisk.c, the /dev/hda2 disk image is loaded into the RAM disk, then the root file system is switched to the RAM disk. The process of refining the disk image continues until everything is “perfect”.

Programming EPROMs

The process of programming (“burning”) the EPROMs starts out by archiving the small disk drive with **tar**, then extracting the files onto a clean (zeroed out) file system. By putting the file system onto a clean disk, all unused sectors are zeroed out, and the disk compression works (Listing 1).

To tar the disk image, the “full ”Linux partition was booted, and the 6MB partition mounted. By doing this, the proc file system is not included in the tar. The following commands can be used:

```
mount -t ext2 /dev/hda2 /mnt
cd /mnt
tar -cpf /tmp/eprom.tar *
```

To create the (uncompressed) disk image, I used a different machine with a 6MB RAM disk and the following commands:

```
dd if=/dev/zero of=/dev/ram count=12288
mke2fs /dev/ram 6144
mount -t ext2 /dev/ram /mnt
cd /mnt
tar -xpf ~/eprom.tar .
dd if=/dev/ram of=~/eprom.dsk count=12288
```

This creates a file (eprom.dsk) which is a sector-by-sector image of the disk. The data to be programmed into the EPROMs is the compressed image. This is done with a program (med.c) which reads the disk image (eprom.dsk), runs the disk compression, and outputs a binary file (eprom.img) which will be programmed into the EPROMs.

```
med ~/eprom.dsk ~/eprom.img
```

The EPROM image is then moved to an EPROM programmer and the images are burned.

DOS boot SSD

Fortunately the SBC came with SSD utilities to help build the disk image. The DOS SSD disk has a bare minimum of files in it: the DOS boot files, command.com, autoexec.bat, the keyboard loading program, loadlin and zImage.

Listing 3

Listing 4

Conclusion

The development of what goes on the disk is a large part of the job, and methods need to be developed to minimize this effort. Using the EPROM disk is working well in our application.

Dave Bennett “works with computers” at Boeing in the commercial Flight Test group. When not at work, he enjoys the company of his significant other, two cats, a bunch of fish and millions of yeasties. Dave enjoys building things, a few

of which are featured on the web page www.halcyon.com/bennett. Dave can be reached at bennett@halcyon.com or dave.bennett@boeing.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Using Linux with Programmable Logic Controllers

J.P.G. Quintana

Issue #33, January 1997

After a short introduction to programmable logic controllers (PLCs), see how Linux and the Web can enhance monitoring and control of mechanical systems using test machines designed by the DuPont-Northwestern-Dow Collaborative Access Team.

When solving control system problems for the “real world”, a toolkit approach to problem-solving often leads to quicker and more robust solutions. This is one of the reasons we are using Linux on commercial Intel-based machines at the DuPont-Northwestern-Dow Collaborative Access Team (DND-CAT) at the Advanced Photon Source (APS). The APS (see http://www.aps.anl.gov/) is one of three third generation synchrotron X-ray sources that will provide the world's most brilliant source of X-rays for scientific research. The DND-CAT (see http://tomato.dnd.aps.anl.gov/DND/) is a collaboration formed by the DuPont Company, Northwestern University, and the Dow Chemical Company to build and operate scientific equipment at the APS to study industrial and academically interesting problems in chemistry, biology, materials science and physics. Linux (like all Unix systems) is designed around the toolkit paradigm. The tools which run under Linux provide an excellent framework for building user interfaces (e.g., Netscape, Java, Tcl/Tk, expect, World Wide Web daemons), running calculations (e.g., C, C++, FORTRAN, Perl, pvm) and interacting with external devices (GREAT access to serial devices, cards in the backplane, and of course, TCP/IP).

However, while there are efforts to equip Linux with real-time capabilities, it is not a “real-time” operating system. In addition, using commercial personal computers for control applications is a mixed blessing at best. While the systems are powerful, readily available and inexpensive, they also come with a limited number of slots on the backplane and the machine usually must be physically close to the process being controlled or monitored. This can be problematic in situations where the process takes place in a harsh environment that might cause the hardware to fail (e.g., high radiation areas, high vibration,

etc.). These are important factors in the design of an entire control system. However, they are only problems if we expect Linux to provide the entire solution to the control problem rather than one tool in a toolkit approach.

At the DND-CAT, we have been designing systems that use programmable logic controllers in conjunction with Linux PCs to provide low cost automation and control systems for scientific experimental equipment.

Programmable Logic Controllers

Programmable logic controllers (PLCs) are the unsung heros of the modern industrial revolution. Long before IBM and Apple were churning out computers for the masses, factories were being automated with computerized controllers designed to interface with the "real world" (i.e., relays, motors, temperatures, DC and AC signals, etc.). These controllers are manufactured by many companies like Modicon, Allen-Bradley, Square D and others. In his booklet, *History of the PLC*, Dick Morley, the original inventor of the PLC, notes that the first PLC was developed at a consulting company, Bedford Associates, back in 1968. At this time, Bedford Associates was designing computer-controlled machine tools as well as peripherals for the computer industry. The PLC was originally designed to eliminate a problem in control. Before the digital computer, logic functions were implemented in relay racks where a single relay would correspond to a bit. However, relays tend to be unreliable in the long term and the "software" was hard programmed via wiring.

System reliability could be improved by replacing the relays with solid state devices. This had the advantage that the system was maintainable by electricians, technicians and control engineers. However, the "software" was still in the hard wiring of the system and difficult to change. The alternative at this time was using one of the minicomputers being developed, like the PDP-8 from Digital Equipment. While more complex control functions could be implemented, this also increased the system complexity and made it difficult to maintain for people on the factory floor.

Morley designed the first PLC to replace relay racks with a specialized real-time controller that would survive industrial environments. This meant that it had to survive tests, such as being dropped, zapped with a tesla coil and banged with a rubber mallet. Designed for continuous operation, it had no on/off switch. The real-time capabilities were—and for the most part still are—programmed into the unit using ladder logic.

Ladder logic is a rule-based language; an example is given in Figure 1. The line on the left side of the diagram shows a "power rail" with the "ground" for this rail on the right hand side (not shown). The rules for the language are coded by completing "circuits" in ladder rungs from left to right. In the diagrams, "||"

corresponds to switch contacts, and "()" corresponds to relay coils. Slanted bars through the contacts and coils denote the complement. The "X" switch contacts are mapped to real binary input points, the "Y" relay coil contacts are mapped to output points, and the "C" contacts/coils are software points used for intermediate operations. In the example, closing X0 and C0 or opening X0 and C0 will energize the C10 coil thereby closing the C10 contact. The C10 contact activates Y0 and turns off Y1.

While this graphical style of programming may be strange for someone accustomed to programming in C or FORTRAN, ladder logic makes it easy for non-programmers to write useful applications. Most PLCs have a large set of functions, including timers, counters, math operations, bit shifters, etc. They have a wide variety of input and output devices, including binary and analog inputs and outputs, motor and temperature controllers, relay outputs, magnetic tachometer pickups, etc. The number of input and output points depends on the type and size of the PLC, but can range from less than 10 for a micro-PLC to over a thousand for one of the higher-end PLCs. The PLC market has grown over the years and has been affected by the computer revolution. Today, there are a number of high quality, inexpensive PLCs on the market. PLCs from the same vendor can often be networked together. In some cases, a lower-end PLC can be built for less than $500.00.

PLCs and Linux

In our applications, we have used Linux computers as interfaces between the PLC and the outside world. Our main purpose for this was to use tools like Tcl/Tk and the World Wide Web (WWW) through the Common Gateway Interface (CGI) to control processes in the PLC. The software used to program our PLCs is Microsoft Windows-based. Consequently, by having Linux and Windows partitions on the same hard disk, we can toggle back and forth between MS Windows, where we program the PLCs, and Linux, where we program and use the operator interface.

The real-time operating system inside PLCs is relatively simple compared to a complex system such as Linux. Consequently, those portions of the control process which require extremely high reliability can be programmed into the PLC, leaving Linux available for other tasks.

We are using the PLC Direct line of PLCs (see http://www.plcdirect.com/) for our applications. In order to prove the reliability of the Linux and PLC Direct combination, we collaborated with the UNICAT (A University-National Lab-Industry Collaborative Access Team; see http://www.uni.aps.anl.gov/) to set up a test system using a Linux-based web server connected to a PLC Direct 405 PLC. Communication with the PLC is through a multidrop, packet-based, master/slave protocol that runs over a serial link. Using the PLC Direct

documentation, we implemented this protocol using Don Libes' expect program over the Linux serial ports, making the Linux system the master. This gave us the capability to “peek” and “poke” into the PLC memory map. CGI scripts call the expect program to provide access to the Web.

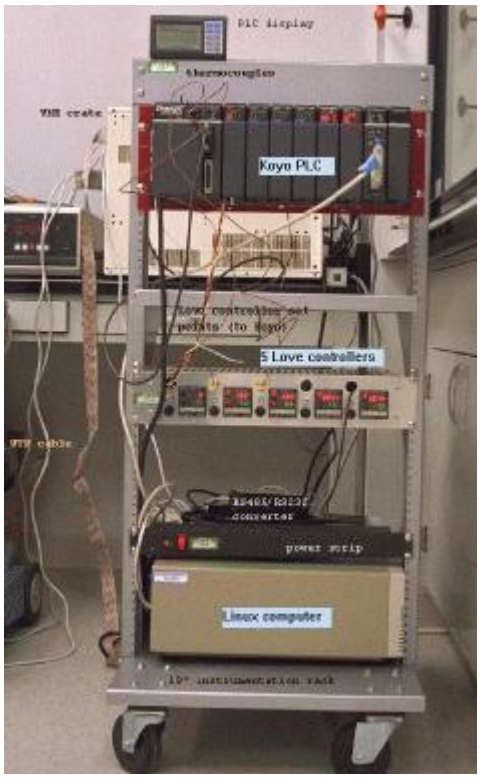
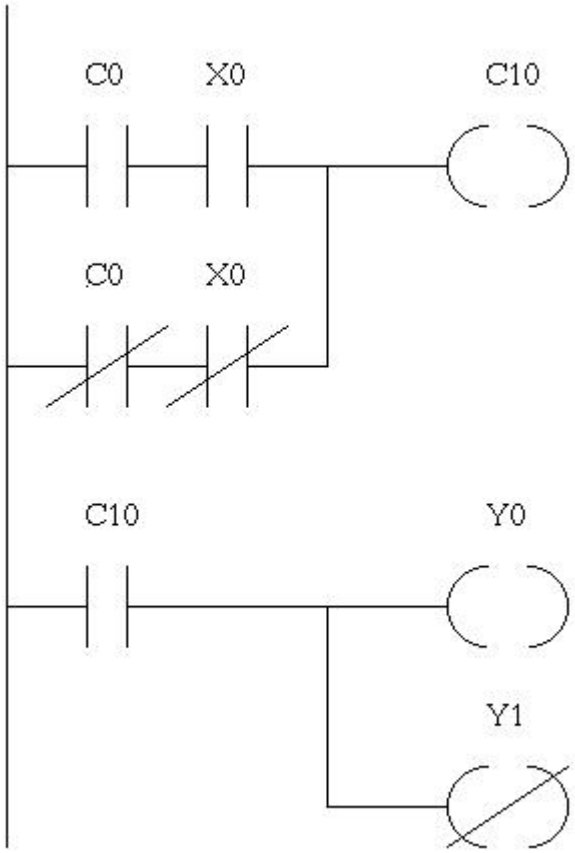


Photo of the DND-CAT/UNI-CAT Linux/PLC Test Stand

Net surfers were allowed to close output points and read input points on the PLC through the WWW interface between March 1995 and July 1996. A photo of that test stand is shown in Figure 2. The PLC monitored digital inputs from the 5 Love controllers, which measured the temperature in the PLC CPU and closed a contact if the temperature went above a preset value.

In addition to the demonstration, we have used the Linux/PLC combination in three project areas: a simple shutter for a synchrotron X-ray beamline, a personnel safety system for an analytical X-ray machine, and an equipment protection system using a high intensity X-ray beamline at the Advanced Photon Source.



Simple Ladder Logic Diagram

Our simplest application with the PLC and Linux involved interfacing a commercial X-ray beam shutter to our Linux data collection computer. The hardware for the X-ray shutter is controlled by two relay-actuated solenoids. When we program the PLC, we allocate two ranges of control relays to act as an interface between the PLC and Linux. The program in Figure 1 demonstrates this. The Linux program that would set C0. X0 is attached to a hardware switch and provides an external input to the system. The X0 and C0 combination simulate a three way switch, and Y0 and Y1 actually operate the relays on the shutter. A program on the Linux side can read C10 to monitor the shutter status. With the interface between the PLC and Linux defined through control relays, the actual control process is divided up between the two different machines.

Our second project used the PLC as a state machine to monitor a radiation enclosure for an X-ray generator and X-ray tube. Since this was a safety device, we enabled the PLC's password function to lock the program into the PLC CPU. If for some reason we forget the password, the CPU must be sent back to the manufacturer for reset. The PLC monitors twelve door contact switches, switches from an operator panel, X-ray shutter positions, water flow interlocks for the X-ray tube, as well as providing a buzzer and a fail safe lamp to notify the operator the X-rays and shutter are on. The PLC also provides enable signals for the X-ray generator and the X-ray shutter.

While the main purpose of the PLC is to protect the operator, the PLC doesn't have a very good way of notifying the operator of what has failed should the X-ray interlock trip. This is where Linux comes in. Using CGI scripts, we wrote web pages that allow the operator to query the PLC state using a browser. To prevent unauthorized access to the equipment (only trained people can use this equipment), we provided a watchdog signal between Linux and the PLC. An authorized user logs into the Linux system and runs a protected daemon which starts the watchdog timer in the PLC. The Linux daemon must continuously restart the watchdog to keep the X-ray system enabled, and the daemon disables the system when the user logs out. Linux keeps track of all of the accesses to the system and sends e-mail to the X-ray generator custodian whenever an access occurs. Thus, the Linux system acts as the data collection computer for the instruments attached to the X-ray generator.

Our last project is an equipment protection system for an X-ray synchrotron beamline at the Advanced Photon Source. In this case, the PLC is monitoring over 70 input points from water flow meters, vacuum system outputs, and switches from vacuum valves. Based on the status of these systems, the PLC sends an enable or disable signal to the APS which permits them to deliver the high intensity X-ray beam to our equipment. Serious equipment damage can occur if the APS delivers beam when the systems are not ready. In this case, we use Linux as a data logger as well as an operator interface. Every few minutes, Linux polls the PLC to log system status.

The PLC itself keeps a log of significant events in nonvolatile memory in the event of power failures. In order to keep the PLC in sync with the Linux logs, we use the Network Time Daemon on the Linux end and once a day reset the real-time clock in the PLC. In addition to the PLC, Linux processes are monitoring other devices, like vacuum gauges, through a multiport serial card. If a system failure occurs, our scientists and engineers can either log into the Linux system and run expect scripts to diagnose the problem or use a browser and interact with the Linux/PLC combo via the Web. At this point, the operator has complete control over enabling and disabling processes in the PLC.

In this application, the interface with the World Wide Web is extremely important. Scientists travel to synchrotron sources from all over the world to conduct experiments. When the facility is operational, it runs twenty four hours a day. If our PLC were to shut the equipment down, it is important to be able to diagnose the fault, and if possible, return the equipment to operational status as quickly as possible. By using the World Wide Web, we provide our scientists and engineers with diagnostic tools they can use from anywhere using commonly available interface software. I personally have been able to monitor system status in our PLCs at my desk at work, my apartment in Chicago and a cyber-cafe in London.

In general, we have found combining programmable logic controllers with Linux to be a cost effective and robust method for providing specialized control systems at the DuPont-Northwestern-Dow Collaborative Access Team. As we build our instrumentation, we continue to find new applications for this combination. We have several more projects in the works, including using the PLCs to construct intelligent controllers for specialized machines and using Linux to interface with them. We also plan to implement the PLC Direct slave protocol under Linux to allow the PLC to send data directly to Linux daemons, so the PLC does not need to be polled.

Acknowledgements and Notices

I would like to thank Pete Jemian of the UNI-CAT for collaborating on the prototype system and for the photo in Figure 2.

The DND-CAT Synchrotron Research Center is supported by the E.I. Du Pont de Nemours & Co., The Dow Chemical Company, the State of Illinois through the Department of Commerce,and the Board of Higher Education Grant IBHE HECA NWU 96 and the National Science Foundation Grant DMR-9304725.

Reference to specific products does not constitute a commercial product endorsement.

References

Libes, Don. *Exploring Expect*, (0'Reilly & Associates, Inc.) 1995.

Morley, R.*History of the PLC*, R. Morley Inc. Milford NH.

Quintana, J.P.G., and Jemian, P. *Design Criteria for Beamline Protection Systems at the Advanced Photon Source*, Synchrotron Radiation Instrumentation Conference 1995 (in press).

John Quintana (jpq@nwu.edu) is an assistant research professor in the Department of Materials Science and Engineering at Northwestern University and is working at the DND-CAT facility at the Advanced Photon Source. In his little spare time, he enjoys aikido, kaledeioscopes, hiking with his wife and petting puppies at the animal shelter. If you have any questions or comments, he can be reached by email or by post at the DND-CAT, Building 432/A003, 9700 South Cass Avenue, Argonne, IL 60439, USA.

Archive Index Issue Table of Contents

Advanced search

Disk Maintenance under Linux (Disk Recovery)

David A. Bandel

Issue #33, January 1997

The ins and outs of disk maintenance—what we all should know and DO.

Here's a hypothetical situation for you to think about. You're working on your Linux box, calling up an application or data file, and Linux hesitates while reading the hard disk. Then, scrolling up the screen (or console box), you see something like this:

```
Seek error accessing /dev/hdb2 at block 52146,
    IDE reset (successful).
```

After some time spent chugging away accessing the drive, Linux continues. If you're lucky, everything is still running along fine. If you're not, your program is refusing to start, or your data file contains garbage.

Chances are, if you're using a hard disk drive that's a few years old, you will begin to see errors when accessing the disk from time to time. At this point, the best prognosis for your disk is that, given time, it'll get worse. So you need to begin resuscitation efforts as soon as possible. Several disk manufacturers have utilities that find and allocate these bad sectors on your hard disk. Unfortunately, these utilities also destroy the information on your disk, and are normally run from DOS, not Linux.

Fortunately, Linux has some system utilities to help you when you are dealing with its (now) native ext2 format. (Utilities are also available for minix. If you need to repair other non-Linux file systems you should use their own native sets of file system utilities.) While not as user-friendly as Norton Disk Doctor or Microsoft ScanDisk, the Linux disk and file system utilities get the job done. In this article, we'll look at a few of the tools to help us overcome the kind of problem I described in the opening paragraph. Other hard disk manipulation utilities can be found in **/sbin** and **/usr/sbin**, but they'll have to wait. For now, let's get the hard disk working properly.

Before you dig in, if you're using one of the newer 2.0.x kernels with an IDE drive, check to see if you have the proper bug fixes compiled into the kernel. If you aren't sure which chipset you have in your computer or are unable to ascertain for sure, it is safe to compile in the CMD640, RZ1000 and Intel 82371 options. These options are found under Floppy, ID and other block devices in your make config. This could save your data in the future. These bug fixes may be all you need, but further checks on your hard drive won't hurt.

I hate cliches, although I'm frequently accused of (ab)using them. If it really went without saying that we always do system backups, my income might be somewhat lower than it is. For most people, it's just not true. So, if you've neglected the chore for a while, just let me say that **now** would be a good time to do that backup. Some of the work I'll be telling you how to do could inadvertently damage or destroy your file system or some of your important files—so be careful and don't say I didn't warn you.

Preparations

Now that I've gotten the requisite legal protection warnings out up front, let's begin. The safest way to start is with a fairly mundane check of the file system. On my system—a combination Red Hat (I like the SYSVinit style bootup), Slackware, Internet tarball concoction—I have fsck, a front-end program that reads the type of file system on a device (from **/etc/fstab**), then invokes the appropriate fsck.filesystemtype checker—in my case, fsck.ext2. You may have e2fsck on your system instead of, or in addition to, fsck.ext2. Don't worry, they're the same file. One may be a soft link to the other, but it's better to make that a hard link.

Before starting, let's prepare our systems for the kind of work we're going to be doing. Whenever I perform low-level maintenance on a system, I find it prudent to ensure I am disconnected from the network. Normally this means dropping to single-user mode. You may opt to do some of these tests from init level 2 (with no network connections), but you'll want to ensure that you don't have too many processes running that want to write to the disk, and none that run from the partition you need to work on. Single-user mode was made for this. A simple **telinit 1** will get us to single-user mode.

If you're not checking the root file system, unmount the file system you're going to work on before you begin. If you forget, you'll get a prompt from fsck telling you the file system is mounted and asking if you want to continue anyway. Say “No”--running low-level system diagnostics, particularly those that alter the file system by writing directly to the disk as fsck does, with the disk mounted, is a very bad idea. Obviously, we can't unmount the root file system. We should be able to remount it as read-only, but a bug in mount doesn't always allow this option. If you need to check the root file system, you can reboot into single-

user mode with the root partition mounted read-only by issuing the **-b** switch at the LILO prompt. The **-b** switch will be passed through LILO to init and will cause an emergency boot that does not run any of the startup scripts. If you have always wondered why you would want to create several partitions—for example, for **/usr** and **/home**--and restrict the size and scope of the root partition, now you know.

fsck—The File System Checker

Invoking fsck from the command line on any given partition will probably not result in a check being run, because you have not reached the predetermined maximum mount count; therefore, the system believes the file system is clean and not in need of checking. To force the check, invoke fsck with **-f**.

At this point, one of two things will happen: fsck will begin to run correctly and check your disk partition (possibly hesitating at the bad spots on the disk and issuing appropriate error messages before continuing) or it will terminate without running, leaving error messages behind. If fsck does not run, you'll have to give the program additional information as indicated in the error messages. Probably the most common information you'll need to pass to e2fsck is the address of the alternate superblock or the block size so that e2fsck can calculate where an alternate superblock is located. The **-b** switch will tell e2fsck to use the alternate superblock, but we'll have to tell e2fsck where to find one. On ext2 file systems, superblocks are normally located at 8193, 16385 and higher multiples of 8192+1 (see dumpe2fs explanation below). As an alternative, we can pass e2fsck the block size with the **-B** switch (once we have that information) to allow e2fsck to calculate alternate superblock locations. Later I'll tell you where to get the block size value if you ever need it.

At this point, it's worth mentioning two other mutually exclusive switches available to fsck and e2fsck. The first is the **-n** switch, which tells fsck to answer no to all queries, and will leave the file system in its original condition making no repairs. The second is the **-y** switch, which automatically corrects any errors it finds. Generally, to speed things up, you may want to run fsck with the **-y** switch. So, why don't we just use this option all the time? I strongly recommend against this course of action, if you suspect problems with the file system. While fsck will usually not encounter problems, typing **fsck -y** and then taking a coffee break, leaving the machine to take care of itself, is not particularly prudent. If, in the interests of speed, you use the automatic answer yes switch to do routine checks, be sure to list your lost+found directories from time to time. Besides, you'll really want to note the block or inode numbers that appear while fsck runs, so that you can check them later to see if they are allocated to files.

The other available options for fsck and e2fsck can be found in the man pages. I consider the fsck and e2fsck man pages fairly well written, as is appropriate considering the importance of these utilities to your file system's health.

Some Common fsck Messages

You may encounter messages asking if you want fsck to correct an error. Answering no will normally terminate the program so that you may fix the problem and rerun fsck. However, most error messages you're likely to encounter are fairly routine, and you may safely answer yes to them. If you see a message such as **inode 1234 unattached**, it means the file pointed to by inode (information node) 1234 has, for one reason or another, lost its filename. This can occur for several reasons, including a power failure or a computer reset without a proper disk sync.

Other common errors include zero time inodes, which are also due to the disk not being properly synced before shutdown. If you see these errors frequently and you've been shutting down your system correctly, you may have any number of other problems. In this case, you could begin by checking your power and data connections and your power supply for fluctuations or passing too much noise. Finally, check your hard disk parameters. I must caution you that altering the default hard disk parameters could do serious damage to your file system or corrupt your files—be careful.

The lost+found Directories

One lost+found directory should be located in the root partition of each file system. If you have, for example, two mounted file systems, **/usr** and **/home**, you should have three lost+found directories. These directories will contain files whose inodes have become disconnected from their file names. The files in these directories will have the form **./#nnnn**, where **nnnn** is the inode number used as the file name. You may be able to determine what the file is by inspecting it using **cat**. If **cat** returns what appears to be garbage, you probably have a binary file. In this case, you can do a **chmod +x #nnnn**, and then run the file. These procedures should give you enough information to learn what the file is. If the file is important, it can be renamed and moved to its original location; otherwise, it can be deleted.

Down in the Dumps

The next utility we'll look at is dumpe2fs. To invoke this utility, type **dumpe2fs *device***, to get the block group information for a particular device. Actually, you will get more information than you're likely to use, but if you understand the physical file system structure, the output will be comprehensible to you. A sample output is shown in Listing 1.

Output from dumpe2fs

We really need only the first 22 lines of output. (The very first line with the version number is not part of the output table.) Most of these lines are fairly self-explanatory; however, one or two could use further explanation. The first line tells us the files system's magic number. This number is not random—it is always **0xEF53** for the ext2 file systems. The **0x** prefix identifies this number as hexadecimal. The **EF53** presumably means Extended Filesystem (EF) version and mod number 53. However, I am unclear about the background of the 53. (Original ext2fs versions had 51 as the final digits, and are incompatible with the current version.) The second line indicates whether a file system is clean or unclean. A file system that has been properly synced and unmounted will be labeled **clean**. A file system, which is currently mounted read-write or has not been properly synced prior to shutdown (such as with a sudden power failure or computer hard reset), will be labeled **not clean**. A **not clean** indication will trigger an automatic fsck on normal system boot.

Another important line for us is the block count (we'll need this later) that tells us how many blocks we have on the partition. We'll use this number when necessary with e2fsck and badblocks. However, I already know how many blocks I have on the partition; I see it every time I invoke **df** to check my hard drive disk usage. (If this were a game show, the raspberry would have sounded.) Check the output of df against dumpe2fs—it's not the same. The block count in dumpe2fs is the one we need. The number df gives us is adjusted to show us only the number of 1024k blocks we can actually access in one form or another. Superblocks, for example, aren't counted. Have you also noticed that the “used” and “available” numbers don't add up to the number of 1024k blocks? This discrepancy occurs because, by default, the system has reserved approximately five percent of these blocks. This percentage can be changed, as can many other parameters listed in the first 22 lines of the dumpe2fs readout; but again, unless you know what you are doing, I strongly recommend against it.

By the way, the information you are reading in the dumpe2fs is a translation into English of the partition superblock information listed in block one. Copies of the superblock are also maintained at each group boundary for backup purposes. The **Blocks per group** value tells us the offset for each superblock. The first begins at one, the succeeding are located at multiples of the **Blocks per group** value plus 1.

While we don't really need to use more than the first 22 lines of information, a quick look at the rest of the listing could be useful. The information is grouped by blocks and reflects how your disk is organized to store data. The superblocks are not specifically mentioned, but they are the first two blocks that are

apparently missing from the beginning of each group. The block bitmap is a simple map showing the usage of the blocks in a group. This map contains a one or zero, corresponding to the used or empty blocks, respectively, in the group. The inode (information node) bitmap is similar to the block bitmaps, but corresponds to inodes in the group. The inode table is the list of inodes. The next line is the number of free blocks. Note that, while some groups have no free blocks, they all have free inodes. These inodes will not be used—they are extras. Some files use more than one block to store information, but need only one inode to reference the file, which explains the unused inodes.

badblocks

Now that we have the information we need (finally), we can run badblocks. This utility does a surface scan for defects and is invoked by typing, as a minimum:

```
badblocks /dev/
```

The *device* is the one we need to check (hda1, sda1, etc.) and the *blocks-count* is the value we noted after running dumpe2fs (above).

Four options are available with badblocks. The first option is the **-b** with the block size as its argument. This option is only needed if fsck will not run or is confused about the block size. The second option **-o**, which has a filename argument, will save to a file the block numbers badblocks considers bad to a file. If this option is not specified, badblocks will send all output to the screen (stdout). The third option is **-v** for verbose (self- explanatory). The final option is **-w**, which **will destroy** all the data on your disk by writing new data to every block and verifying the write. (Once again, you've been warned.)

Your best bet here is to run badblocks with the **-o** filename option. As bad blocks are encountered, they will be written to the file as a number, one to a line. This will be very helpful later on. In order to run badblocks in this way, the file system you are writing the file to must be mounted read-write. As root—and you should be root to do this maintenance—you can switch to your home directory, which should be located somewhere in the root partition. badblocks will save the file in the current directory unless you qualify the filename with a full pathname. If you need to mount the root partition read-write to write the file, simply type: **mount -n -o remount,rw /**.

Once you have your list of bad block numbers, you'll want to check these blocks to see if they are **in use**, and if not, set them as **in use**. If a block is already marked **in use**, we may want to clear the block (since the data in it might be corrupted), and reset it as **allocated**. Print the list of bad blocks—you'll need it later.

Enter debugfs

The final utility we will discuss is probably the most powerful and dangerous. With debugfs, you can modify the disk with direct disk writes. Since this utility is so powerful, you will normally want to invoke it as read-only until you are ready to actually make changes and write them to the disk. To invoke debugfs in read-only mode, do not use any switches. To open in read-write mode, add the **-w** switch. You may also want to include in the command line the device you want to work on, as in **/dev/hda1** or **/dev/sda1**, etc. Once it is invoked, you should see a debugfs prompt.

We'll be looking at only a limited set of commands for the purposes of this article. I would refer you to the man pages, but the page for debugfs located on my system is out of date and does not accurately reflect debugfs' commands. To get a list, if not an explanation, at the debugfs prompt type **?**, **lr** or **list_requests**.

The first command you can try is **params** to show the mode (read-only or read-write), and the current file system. If you run this command without opening a file system, it will almost certainly dump core and exit.

Two other commands, **open** and **close**, may be of interest if you are checking more than one file system. **Close** takes no argument, and appropriately enough, it closes the file system that is currently open. **Open** takes the device name as an argument.

If you wish to see disk statistics from the superblock, the command **stats** will display the information by group.

Now that you've had a chance to look at a few of debugfs' functions, let's get to work fixing our hard disk. From the printed list of bad blocks, we need to see which blocks are in use and which files are using them. For this we'll use **testb** with each block number as an argument. If the test says the block is not in use, we know we have'nt lost any data here yet.

If the block is marked as in use, you'll want to find out which file is using this block. We can find the inode by using:

```
icheck
```

which will return the inode that points to the block. From here, we can use

```
ncheck
```

to get the name of the file corresponding to the inode. Now we finally have something we can work with. You may want to try to save the file, but if the

block really is bad, you're probably better off reinstalling this file from a backup disk. To free the block, you can use one of several commands; the one I recommend is:

```
cleari
```

This will deallocate the inode and its corresponding blocks. Remember, you'll have to be in read-write mode to do this. Note that these commands are irrevocable in read-write mode.

Once the bad block has been deallocated, you can use:

```
setb
```

to permanently allocate the block, removing the inode that points to it from the pool of free inodes.

That's it. Once the appropriate changes have been made to set the blocks, you can quit debugfs and reboot. You should not see more problems unless you missed a block (or have grown more bad blocks).

Summary

Good disk maintenance requires periodic disk checks. Your best tool is fsck, and should be run at least monthly. Default checks will normally be run after 20 system reboots, but if your system stays up for weeks at a time as mine often does, you'll want to force a check from time to time. Your best bet is performing routine system backups and checking your lost+found directories from time to time. The dumpe2fs utility will provide important information regarding hard disk operating parameters found in the superblock, and badblocks will perform surface checking. Finally, surgical procedures to remove areas grown bad on the disk can be accomplished using debugfs.

David Bandel is a Computer Network Consultant specializing in Linux, but he begrudgingly works with Windows and those “real” Unix boxes like DEC 5000s and Suns. When he's not working, he can be found hacking his own system or enjoying the view of Seattle from 2,500 feet up in an airplane. He welcomes your comments, criticisms, witticisms, and will be happy to further obfuscate the issue. You may reach him via e-mail at dbandel@ix.netcom.com or snail mail c/o *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Satellite Tracking with Linux

Kenneth E. Harker

Issue #33, January 1997

Looking for something fun to do with your Linux box? One of the most impressive applications available for Linux is SatTrack 3.1 for Unix.

Looking for something fun to do with your Linux box? One of the most impressive applications I've seen available for Linux is SatTrack 3.1 for Unix by Manfred Bester. SatTrack is a very powerful tool for satellite tracking and orbit prediction. It can tell you when a satellite, the Mir Space Station, or a space shuttle will be overhead, where they are at any time and when you might be able to see them. It can even control some external tracking hardware, such as antenna rotors or telescopes. The program can run from the console or a terminal window, or if launched inside an xterm, it can bring up a world map showing the current position of any satellite or group of satellites. It can do all this, and best of all, SatTrack 3.1 is free for private, non-commercial use.

Figure 1. SatTrack v3.1 Menu

As an amateur radio operator, I first became interested in using SatTrack to track the many amateur radio satellite in low-earth orbit. These satellites receive an amateur radio signal on one frequency, amplify it, and retransmit it on another frequency. Two stations on opposite sides of a continent or an ocean can communicate through one of these satellites using radios that ordinarily wouldn't be able to reach each other. Since most of these satellites are in low earth orbit, however, they pass overhead infrequently, and then only for several minutes at a time. Knowing when they will appear where is imperative to successful communication. To solve this tracking problem, it's Linux and SatTrack to the rescue!

Figure 2. SatTrack v3.1 Multi-Satellite Line Display

SatTrack was designed to compile and run on most Unix operating systems, and it runs quite nicely under Linux. One really nice feature of the program is

that it can operate from a terminal window, such as a VT100 terminal, or from a Linux virtual console, if you can't or don't want to use X. On the other hand, if you do run X, you can pull up a full-color world map that will show you where any satellite or group of satellites is at the moment, where the sun is, where the gray line separating the sunlit side of the planet from the dark side of the planet is, and a graphical projection the path of the next several orbits of a satellite. Tools that come with SatTrack can automate the process of downloading fresh Keplerian two-line elements—the files that describe a satellite's orbit—directly from the Air Force Institute of Technology's FTP server.

Figure 3. SatTrack v3.1 Graphic Tracking Display

SatTrack is widely used in the space community. According to the program's home page:

> Users of SatTrack V3.1 include NASA's Jet Propulsion Laboratory (JPL), Johnson Space Center (JSC), Goddard Space Flight Center (GSFC) and Marshall Space Flight Center (MSFC), the European Space Agency (ESA), the German Aerospace Research Establishment (DLR), the National Aerospace Laboratory (NLR) of The Netherlands, the Los Alamos National Laboratory (LANL), the Amundsen-Scott South Pole Station, a large number of universities and other institutions, as well as thousands of private users all over the world.

You can find more information about SatTrack, including its copyright and licensing information and the latest version of the SatTrack 3.1 distribution at http://www.primenet.com/~bester/sattrack.html.

Compiling SatTrack in a Linux system is fairly straightforward. Instructions on how to compile the program are included in the file SatTrack/src/README_MAKE, and I won't repeat all of them here. The most important options to choose, though, are in SatTrack/src/Makefile and SatTrack/src/include/sattrack.h. (These will make sense if you look at the files in question.) In the Makefile, the machine type I chose is (yes, I actually do have a 486)

```
# i486          (i486 with Linux)
#
CPU             = i486/Linux
CC_CMACH        = -O2 -m486
CC_LMACH        =
CC              = gcc
```

The user compile-time options I chose were:

```
CC_CUSR     = -DREVERSEVIDEO -DSUNTRANSITS
-DXWINDOW
```

I used the following options for compiling and linking:

```
X11         = /usr/include/X11
LX11            = -L/usr/X11R6/lib -lX11 -lXt
-lICE -lSM
# X11R6
```

In SatTrack/src/include/sattrack.h, the only changes I made were to define the home directory:

```
#define SATDIR    "/usr/local/bin/X11" /* directory where SatTrack */
```

and to change the print command (you might choose to use nenscript or some other ascii to postscript utility):

```
#define PRINTCMD              "a2ps"
#define PRINTOPT              "-nL -nu -c"
```

After these changes, SatTrack should compile successfully. One final thing to check after compiling is the write privileges on the SatTrack/pred/ directory; if you install SatTrack in a common bin directory, you might want to make that directory world writable or SatTrack may not be able to write an orbit prediction to a file (or print it). For simplicity in calling the program, I've defined a choice in one of my pull-down fvwm menus as follows:

```
        Exec    "SatTrack"      exec
/usr/bin/X11/color-xterm -sb -sl 500 -j
-ls -fn 7x14 -geometry 132x26 -T 'SatTrack
for Unix 3.1' -n 'SatTrack' -e
/usr/local/bin/X11/SatTrack/run/sattrack &
```

Using SatTrack is straightforward. After telling the program your location and the satellite you're interested in, you are presented with a list of options. You can compute an orbit projection over several days. SatTrack will succinctly describe every pass over your location including times and durations of each pass, the peak elevation the satellite will reach, and whether it will be visible. Another option is to display the current position of the satellite. You can display one satellite at a time or a whole group of satellites (such as all the amateur radio satellites). SatTrack will tell you which satellites are currently over your horizon and how long it will be until satellites enter or leave your field of view. When appropriate, SatTrack can even tell you the radio frequency uplink and/or downlink of a satellite, and dynamically compensate for the Doppler shift! More advanced users can integrate SatTrack with antenna or telescope pointing hardware to track an object precisely as it passes overhead. SatTrack is on of very few satellite tracking program commonly available for any operating system that can show satellite visibility or actively control tracking hardware.

The following are SatTrack Stellite Icons

U.S. Space Shuttle Icon

Space Shuttle Icon

MIR Icon

International Space Station Icon

SatTrack is an absolute wonder for tracking satellites. With the help of SatTrack, I've made my first two-way communication with another ham radio operator over 1000 miles away using the Russian amateur radio satellite RS-15, and I look forward to making many more. Those who are not ham radio operators can find fun uses for SatTrack as well. Weather satellites, geological survey satellites, and others are constantly sending image and telemetry data to the earth. SatTrack can also tell you when satellites might be visible. Ever seen a space shuttle fly overhead? The Mir Space Station? The Hubble Space Telescope? With binoculars or a telescope, you might even see smaller satellites, once SatTrack has told you where and when to look. Doing a presentation on the Global Positioning System? Show your audience exactly where the GPS satellites are in orbit. Need a compelling computer display for that sci-fi epic you're producing for Film Studies 101? SatTrack can provide ever-changing graphic disply windows.

Information about SatTrack

Information about SatTrack for Unix is available at http://www.primenet.com/~bester/sattrack.html.

SatTrack 3.1 can be downloaded and used free for private, non-commercial purposes, and the author, Manfred Bester, offers version 4.0 of SatTrack with more options and a more graphical interface for commercial use.

Figure 5: SatTrack Orbit Prediction Short Form

Figure 6. SatTrack v4.0.1 GUI for Multi-Satellite Line Display

To get a taste of using SatTrack to predict times of visibility in a satellite's orbit, check out http://ssl.berkeley.edu/isi_www/satpasses.html.

For more information about amateur radio, see the home page of the American Radio Relay League: http://www.arrl.org/.

For information about Amateur Radio satellites in particular, see the home page of AMSAT, the National Amateur Radio Satellite Corporation: http://www.amsat.org/.

For information about the Shuttle Amateur Radio Experiment, see http://www.nasa.gov/sarex/sarex.html.

Ken Harker, N1PVB has been a licensed radio amateur since October,1993, and an avid Linux user since June 1995. He's currently a graduate student in the Computer Sciences at the University of Texas at Austin.

Archive Index Issue Table of Contents

Advanced search

Free SCO OpenServer Has Its Place

Evan Leibovitch

Issue #33, January 1997

Is Free SCO OpenServer free in response to Linux? Perhaps, in part, but the two operating systems aren't really in competition with each other.

The SCO package looks surprisingly like a typical Linux distribution.

One CD-ROM, two boot floppies, and a leaflet tucked inside a compact package. Primary support is on the Internet, using the Web, newsgroups, FTP and specialized mailing lists (even Compuserve)—Lots of support for PC peripherals. The media costs $19 (U.S. and Canada) plus shipping, but once you have it, you can pass it around to everyone in turn and they can load it for free.

It isn't until you spot that familiar blue tree logo that you know this Isn't really your typical Linux distribution; it's from the Santa Cruz Operation (SCO), the company that's shipped more official Unix servers than anyone in the world.

What you have in your hands is one of the free copies of SCO OpenServer, which SCO has been dishing out by the scores since its introduction at the SCO Forum conference in mid-August 1996. This software is neither crippled nor time-bombed, and it includes a full software development kit. Everyone who installs the free version must register with SCO, but such registration is free of charge, and is done on a web page. SCO says they're issuing free registrations at the rate of about a thousand per week, more than two-thirds of them going to people describing themselves as “technical home users”.

On a different web page heralding free OpenServer to the world (http://www.sco.com/Products/freeopen.html), SCO boasts, this “bold move has far-reaching implications for the future of Unix systems, and marks the stunning public debut of SCO's stewardship of the Unix system.”

Lesser words have made the blood of many a Linux advocate boil, but the “stewardship” is no joke. Since obtaining UnixWare and Unix source code from

Novell last year, SCO owns the AT&T pedigree Unix. Like it or not, SCO is now one of the central players who will determine the success of commercial Unix in the years to come.

Who are the other players? Sun certainly, Hewlett-Packard on sunny days, IBM SGI and DEC during full moons, and, of course, everyone in the Linux community. Given the sheer number and enthusiasm of Linux users, Linux will undoubtedly shape the future of the Unix market, even without the positive effect of Lasermoon's work at Unix certification.

Indeed, the evolution of Linux has already had a major effect on the commercial Unix marketplace. Commercial implementors, like Caldera, have done an excellent job—in a relatively short period of time—of bringing Linux to the attention of the IS community. But they certainly aren't at SCO's level yet—nowhere near.

While the introduction of the free SCO can be traced in small part to the rise of Linux, there are other factors about this release which are more important to SCO. Squeezed by Microsoft on one side and Sun and additional RISC vendors on the other, SCO must be seen as an active player—releasing a free version of its bread-and-butter can certainly be seen as “active”.

Is free OpenServer an SCO assault on Linux? Only a little, since, for one thing, there are strings attached to the software that SCO sells at many times the cost of the most expensive Linux distribution. The main restrictions are a two-user-only license (not upgradable to more users) and a prohibition on commercial use.

In other words, you can set up a web server with the free OpenServer, but you can't sell space on it. You can write and compile all the software you want, as long as you don't try to sell the result. You can run it at home all you want (as long as you don't have a home-based business), but if you want to use it legally at work you'll have to buy the full-priced version.

SCO's goals with free OpenServer (soon to be followed, we are told, by a similarly free version of UnixWare) can be summed up in three words: exposure, excitement and respect.

The exposure part is easiest to explain. SCO wants as many people as possible to get their hands on OpenServer, to get a taste of it, to evaluate it, to learn it and to develop with it. They want SCO Unix in colleges and universities, so the graduates of today will remember SCO when they make the purchasing decisions of tomorrow. They want people installing OpenServer on their home

systems, to increase the level of SCO knowledge in the computer-using populace.

In addition, SCO hopes analysts and other evaluators will take advantage of the free OpenServer, to use it and, it is hoped, dispell some of the many rumours floating around about it. From reading some posts on the Internet, one might surmise the OpenServer of 1996 is merely a dressed-up version of Xenix on steroids.

Now, don't get me wrong. I'm not personally that fond of OpenServer; when my company sells SCO products, it's UnixWare we ship out the door, but SCO's flagship line (that has out-sold UnixWare by a wide margin) doesn't deserve much of the abuse I've seen heaped on it. The free offer is mainly designed to let people see the product and separate fact from fiction.

One of the biggest misconceptions SCO needs to overcome is that OpenServer is a difficult porting target for freeware. While earlier versions of SCO Unix were notorious for their built-by-Microsoft compiler that was a porting nightmare, newer releases have dumped that environment for a considerably more developer-friendly SDK. It's a constant thorn in the side of SCO that so much freeware takes so long to get ported to SCO, and much of it is not ported at all.

All these new copies of OpenServer in the hands of the people, and all these development systems begging for freeware to be ported to them are what SCO hopes will build some excitement around its product.

After all, SCO has historically been a company with good, but boring products. SCO's skill is in blending different (and often hostile) technologies together. However, SCO doesn't produce major inventions like Sun does, doesn't move forward at the frantic pace of Linux, and doesn't do hardware like HP and other Unix vendors.

Attention from the press or public is hard to come by for a product that sells well, but contains very little whiz-bang. And as a result SCO doesn't get respect from independent software vendors. While advances in PC hardware, such as SMP and PCI and the Pentium Pro, allow SCO-based systems to scale well into the turf of the RISC vendors, there are still many enterprise ISVs who ignore SCO despite its huge market share. High-profile developers, like Computer Associates, continue to treat SCO as an also-ran despite the fact that its installed base of Unix servers is larger than anyone else's.

To a certain extent, this can be seen as a lack of respect, a feeling (justified in my opinion) that SCO is the Rodney Dangerfield of the computer industry. By spreading free copies of SCO around the Unix community, SCO hopes to gain

from the analysis by the independent software vendor communities the respect it already has from its users.

Will free availability of SCO's operating system give it the whiz-bang it craves? Hard to tell. Will any new-found respect come at the expense of Linux? Not likely.

It's doubtful most current users of Linux are going to reformat their hard disks to install free OpenServer. No one who runs an ISP, uses his Linux system for writing commercial software or needs more than two users can use it legally for those purposes. And the many people who use Unix at work and also run a version at home for learning purposes are satisfied with Linux performance on minimal hardware.

The free OpenServer (or any OpenServer, for that matter) doesn't come with source code; this limits its value in educational and hobbyist settings, since you can't really tinker with it. And, while the SCO-supported SkunkWare CD is chock-full of freeware compiled for use with OpenServer, SCO still lags far behind Linux when it comes to freeware availability.

Still, the other side has its advantages. SCO still supports far more commercial applications than Linux, though the number of instances where such apps can be run on free OpenServer (legally) will be quite few. Since SCO is installed on so many corporate Unix servers, the free OpenServer will attract those who perceive SCO familiarity as more marketable than Linux in the IS world. Until Linux makes significant inroads into corporate servers and is supported by major database ISVs, this perception will remain a powerful incentive for people to install the free SCO rather than Linux.

So while the free SCO probably won't do much to shrink the existing installed base, it is reasonable to ask what effect it will have on future growth of Linux. Will anyone actually use free OpenServer to set up a personal web site, endure all the cost and trouble of a full-time link and the necessary administration, while being restricted from taking money to put up pages for others? Some of the biggest growth areas for Linux, such as providing corporate Internet (and Intranet) servers, are not threatened by the non-commercial license of free SCO.

There are still some things SCO does better than Linux; official support for major DBMS systems and very high-end hardware are but two. People needing features like these are not considering Linux anyway, so little will be lost if or when these people install Free SCO.

Are there those amongst the body of Linux users who are only using it until "something better" comes along, or haven't installed **any** Unix at home because they didn't want Linux, but would use SCO within its legal parameters? Some at SCO seem to think so.

To me, these numbers are insignificant. The Internet is filled with stories of people who began with Linux because it was all that was available at the cost, then happily realized it performed beyond expectations. While there are few hard facts, there is much anecdotal evidence that Linux will run faster than SCO on a given hardware platform. Certainly Linux will run on older and cheaper hardware better than SCO will. While good new hardware is inexpensive, people getting their OS for free for home use probably won't want to spend much on hardware upgrades.

Still, the biggest distinction to be made between the two products has to do with their ultimate purposes. SCO intends its free Unix to boost sales of that which is not free; regular retail for OpenServer ranges from $1,295 to more than $13,000. SCO is even hoping the free two-user OpenServer will encourage people to pay $795 to use the same product legally for commercial use (the SDK is another $395).

Make no mistake about it—the primary targets of free SCO are Solaris and NT, not Linux and other freeware operating systems. In those goals, it is in the interest of **all** in the community who support Unix on Intel systems (which includes Linux supporters) to encourage SCO's efforts. It is unreasonable to consider these efforts as competition—Microsoft poses a far greater challenge to Linux and SCO collectively than either pose to each other.

This is borne out by my own experiences. Authorized to sell both Caldera and SCO products, my company has not yet come across a situation where both Linux and SCO would be a good fit—either one or the other is best for any particular job. Except for SCO's attempt at an Internet server (which is not available under the free license), there's far less overlap than you may think.

For some, mostly those who already use SCO at work (or hope to), it makes more sense to install the free version of SCO rather than Linux. As a tool to gain attention, free SCO has already succeeded. Respect will be harder to obtain, but it appears to be an attainable goal.

Still, by and large, there will be no major effect on the Linux community because of one persistent difference between the two: Free SCO software is designed ultimately to be just a stepping-stone to the high-priced spread. Linux has no such path to follow, no restrictions to deter, no upgrade needed when

going from experimenter to implementor to commercial administrator. In this regard, free SCO never really was competition to Linux.

Evan Leibovitch is Senior Analyst for Sound Software of Brampton, Ontario, Canada. He's installed almost every kind of Unix available for Intel systems over the past dozen years, and this year his company became Canada's first Caldera Channel Partner. He can be reached at evan@telly.org.

Archive Index Issue Table of Contents

Advanced search

Interview: Caldera's Bryan Sparks

Phil Hughes

Issue #33, January 1997

Where does Caldera go next? *LJ* Interviewed their President and CEO Bryan Sparks to find out.

I had an opportunity to talk to Bryan Sparks at the Unix Expo trade show in New York on October 9. In the past, Caldera has offered a different sort of Linux to the market—one aimed at the “professional” end user. In other words, people who want to use applications, and who just happen to be running a Linux machine. But, with market penetration less than expected, Caldera has made some changes.

On October 7, Caldera announced the release of their Solutions CD, which includes software from Caldera and their Independent Vendor Partners (IVPs). This release offers a set of applications for the Linux market while giving the IVPs access to the global Linux market. The CD contains software that can be released by getting an access key from Caldera. See the sidebar on page FIXME for a list of the products available on this CD. Most of these products are not specifically licensed for use only on Caldera Linux, which means that Caldera is bringing some serious applications to the general Linux community.

Caldera's second announcement was the licensing of Novell's Cross-Platform Services for use on Linux, allowing Caldera Linux platforms to work in a fully integrated fashion with Novell Netware and IntranetWare systems. Other licensees of this technology include Hewlett-Packard and The Santa Cruz Operation (SCO).

Three New Products

Finally, in order to better address the needs of different market segments, Caldera is introducing three Linux-based systems targeted at different market segments. These new products are collectively called Caldera OpenLinux (COL) and are based on the Linux 2.*x* kernel. The effort to produce this integrated set

of products started with LaserMoon who did the first work toward X/Open and other certifications. Caldera is now working with Linux System Technologies of Erlangen, Germany to complete the integration of Caldera and LaserMoon's work with additional technologies.

Bryan and I talked about “their” technologies vs. the standard development paths. Bryan assured me Caldera's intent was to make any necessary changes for POSIX certification and Unix branding available to the Linux community as a whole. He sees Caldera's products as part of the total product mix for the Linux community and wants to make sure Caldera's work continues to be part of the mainstream.

The low-end product, called COL Base, is much like the original Caldera Network Desktop. The big changes are the elimination of a Netware (IPX) client and a change in price from $99 to $59. It will also include the Caldera Solutions CD. Bryan said he thinks this will be a better fit for the casual user—a Windows-like desktop environment at a price more in line with other, plainer, Linux distributions.

The second product, COL Workstation, will be COL Base plus Netscape Navigator 3.0 Gold, Netware Client and Administration, a commercial, secure web server and other technologies yet to be announced. This product will retail for less than $300.

The final product is COL Server. It will include the features of COL Workstation plus Cross Platform Services and GroupWise technologies licensed from Novell. It will be fully capable of interacting as a secure server in an environment that includes NetWare, Unix and Windows NT systems on an intranet or the Internet. This product will offer an alternative to Microsoft's NT Server and will retail for less than $1,500.

COL Base should be available by the time you read this article. The other products will be introduced throughout 1997, with upgrade options available for current Caldera users.

I also asked about support. Caldera includes their own Internet-based support for all products, and will include 30-day installation support on the workstation and server products.

Caldera has over 200 resellers under contract. In order to qualify for the reseller program, the resellers must have Unix training, so that Caldera is assured that they can support the products they sell. Bryan said many of these resellers have been resellers of SCO Unix or UnixWare.

I asked Bryan if Caldera intended to continue with Unix branding of their product. (To be able to use the Unix name, your product must be certified by X/Open.) The answer is yes, and he expects this to happen in 1997. Bryan wants to make sure Caldera does this right, getting any required patches back into the mainstream Linux kernel so everyone will benefit from their work.

What This Means for Linux

Right now Linux is seeing substantial use as a system for connectivity, including web servers, Internet Service Providers and gateway systems to connect office networks to the Internet. With Caldera's new products, it is going to be much easier for companies to put these systems together. This ease of use saves time for the Linux-literate who want to get a system up, and makes it possible for the newbie to buy an answer off the shelf. This means there really is an answer to Windows NT.

Caldera Solutions

Phil Hughes is publisher of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Netactive SynergieServer Pro

Jonathan Gross

Issue #33, January 1997

Every machine is put together by an engineer who approves it for shipping. Before it is shipped the same engineer writes a letter to the recipient explaining the configuration and highlights of the system.

- Product Description: Pentium Pro Server machine.
- Netactive Systems, LTD.
- Price: $3,472 (as of August 5, 1996)
- Reviewer: Jonathan Gross

Sitting on my desk is a 100mhz 486DX-4 workstation with 32M of RAM. It has cables falling out of the case, the cover is just sitting on the floor, the wiring is a mess, and god only knows where the manuals for the components are. It took me at least two days to get all the pieces working together, X configured, and everything running relatively smoothly—and probably cost me around $2,000.

My DX4-100 is relatively fast; I can run a couple copies of XEmacs on it, and a kernel compile takes around twelve minutes. Not too bad, and certainly better than the 386-25 sitting in my closet next to it, or the 486-33 in the living room.

The DX4-100 was enough until something happened at work.

A company called Netactive Systems, Ltd. sent *Linux Journal* a machine to try out—their SynergieServer Pro. My DX4-100 isn't so fast anymore.

I am impressed with Netactive. Usually when you buy a machine, either from a local vendor or mail-order, it is a fairly sterile transaction. You hand them your credit card and they blithely hand you a box full of electronics—it's kind of like going to the Motel 6 of computer stores. Netactive is more like going to the Four Seasons Olympic Hotel to buy a machine.

Every machine is put together by an engineer who approves it for shipping. Before it is shipped the same engineer writes a letter to the recipient explaining the configuration and highlights of the system. An excerpt from ours:

> "Fundamentally, it does not take much engineering excellence to put together a great $15,000 dollar Intel-based computer; just buy a couple of everything with a great big name on its shiny new box and throw it together. It does, however, take a little bit of know-how to make a truly great $3,500 machine..."

The letter goes on to explain that they put 64M of RAM and a nice video card into it instead of SCSI Wide because Caldera (the installed Linux) is a GUI environment, and the RAM and good video card are going to be a better use of your money than faster disks.

This is cool. It gives you a much better idea of what is in the machine, and why (and even that there is a "why"!) than the usual random checklist of components does. The random list of components is also included, as is a *Sysadmins Configuration Guide*, which lists all the configuration settings, from IRQ and hardware addresses to domain name and host name of the machine to the disk partitioning. Both the list of parts and the *Sysadmin Configuration Guide* are sent loose with the other paperwork, and a second copy is taped to the inside of the case. The other paperwork includes: a warranty certificate (three years limited parts, and five year limited labor), the build sheet, and an invoice, all with customer numbers, job number, and serial number for technical support calls and tracking information.

Hardware

In the machine is:

- **Motherboard**: Asus P/I-P6NP5, Intel Nanoma (440FX) chipset with 4 PCI slots, 1 PCI/Media Bus, 3 ISA slots, Optional Infra-red port, and 64MB FPM DRAM, and an Intel Pentium Pro 200 Mhz chip (with a bearing heat sink and fan).
- **Drives**: Asus PCI-SC2000 Fast SCSI-II HDD (NCR53c810), with a 2.1 GB Seagate/Conner Fast SCSI-II, Teac CD56S 6x CD-ROM.
- **Video**: Diamond Stealth 64 Video 2001 with 2 MB DRAM, and a Princeton EO15 15" monitor (1280x1024@70Hz, non-interlaced).
- **And more**: 3Com 595-T4 (10/100 MB/sec Ethernet), ESS 1688 16-bit Soundblaster compatible sound card, Logitech mouse, 3.5" floppy drive, Seagate TapeStor T-3200, 3.2 GB Travan Drive, and USR v.34 internal modem, and, of course, all the manuals and documentation for everything neatly packed in Manila envelopes.

This all came in a **very** nicely configured mid-tower case with a dual fan and 250W power supply with all wires neatly routed and tied off.

Software

Software pre-installed included:

- Caldera Network Desktop v1.0
- Caldera InterNet Office Suite v1.0
- X-Inside Accelerated X Server v1.2
- Red Hat 3.0.3 “Picasso” kernel 2.0.10
- Windows NT
- Dual Booting under NT Boot Loader

I fired it up, and after figuring out how to work the NT boot loader (which I despise), Linux was all there, configured and running the way it should be running. I never booted NT, but I assume it was configured just as carefully.

NT brought up the one thing I didn't like about this machine. I would rather use LILO to boot Linux, and not have NT on the drive at all—I'm sure this configuration could be requested if you too want a Linux-only set-up.

The system was fast—we ran the informal JonGrossSpeedTest (building a kernel). Note that both machines had enough RAM so that swapping wasn't an issue...

Jon Gross Machine Speed Test results:

For kernel version 2.0.10 on the Netactive machine (timed using **time -v**):

```
  Command being timed: "make zImage"
    User time (seconds): 246.31
    System time (seconds): 26.29
    Percent of CPU this job got: 93%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 4:50.37
```

For kernel version 2.0.10 on the DX4-100:

```
  Command being timed: "make zImage"
    User time (seconds): 881.37
    System time (seconds): 117.16
    Percent of CPU this job got: 93%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 17:49.15
```

Netactive has done a hell of a job putting together a computer system that is fast, neat, clean and would be hard to build for the same price by buying parts off the shelf.

Jonathan Gross is a Perl hacker and wants some faster machines. Donations can be sent to info@linuxjournal.com.

Archive Index Issue Table of Contents

Advanced search

Java and Client/Server

Joe Novosel

Issue #33, January 1997

So you think client-server programming is only for large applications?

Client-server applications are everywhere. Client-server can be defined as a process which provides services to other processes. The client and server can be run on the same machine or on different machines on opposite sides of the world. A non-programming example of a client-server situation is the telephone system. You are the client (or customer) and the central office is the server. By having a telephone connected to the system (and your bill current!) you are subscribing to the services that the central office provides. Requests are made of the server (central office) to place and receive calls. The server also does accounting on each call made or received and handles emergency (911) requests. In this article I will present a simple CB (citizen's band) radio simulator which was written for a class project. The server is written in C and the client is written in Java. I will assume that the reader understands what sockets are and has a rough idea of how they are used.

The specifications for the project were loosely:

- Provide a server which can accept multiple simultaneous connections. The server should have a basic command set that will open and close the connection, change the channel and provide a list of current clients on a specific channel.
- Provide an “emergency channel” (9) that will broadcast all traffic to all currently connected clients regardless of which channel they are subscribed to.
- Client should “come up” on channel 19. This is the channel where CBers meet. They then agree which channel to move to in order to continue their conversation.
- The client should display all traffic on the current channel with the handle of the person who sent the message.

- The server can be either a single process concurrent or multiple process server (more on this later).

Why Java?

Besides getting extra credit for doing a graphical user interface, I chose Java in order to simplify programming of the client. Java is used specifically for the following reasons:

- **Portability**. I was developing code at home on my Linux system and running the code on Suns at school.
- **Functionality**. Not just for web page animations, Java is a very useful programming language. Java is object-oriented and very similar to C and C++. A simple user interface is relatively easy to write.
- **Threads**. Java allows multiple threads (like a background process) of execution. A thread can be launched that will listen for incoming messages. When a message comes in, it is automatically sent to output. We start this thread and forget about it.
- **Can be run remotely from Netscape (or similar browser)**. While I have not currently implemented this feature, conceivably, web surfers looking at your page could talk to each other using the CB simulator. There are many restrictions to this which we will go into later.

Why Not C?

Using C for the client would require more programming to accomplish the same results. First, some sort of GUI builder like Motif or X-Forms would need to be used. I'm not knocking any of these, but not every system has them and they can be difficult to learn and use. C does not have threads, so all I/O would have to be polled. User input as well as incoming messages would have to be polled and processed accordingly. Without a GUI, some type of command codes would have to be developed for the user to control the client and server. This would probably be very cryptic and difficult to use—not to mention difficult to implement.

I developed the server first, from the specification in Table 1. Messages are fixed length and must not vary from the given format. C handles transmissions well through the use of structures and pointers; basically, you just call a write or read routine, passing a pointer to the data structure, and the bytes come or go without much of a problem. This works fine for C; Java is another story.

A Few Words on Sockets

Sockets work almost the same in Java as their counterparts in C. Since Java is object-oriented, you must create an instance of the socket object. This is done by a simple line of code:

```
Socket s = new Socket(hostName,portNumber);
```

where **s** is the instance of type **Socket** and **hostName,portNumber** are the name of the host and port to connect to. But a socket by itself is not very useful without a data input and data output stream. The code segment below sets up a data input and output stream to talk to the socket:

```
dis= new DataInputStream(s.getInputStream());
dos= new DataOutputStream(new
BufferedOutputStream(s.getOutputStream()));
```

The output stream is created as a buffered output stream. Data will not be written across the socket unless either Java feels that there is enough data to write, or you force a write—using a flush by using something like: **dos.flush()**; this calls the flush method on the data output stream. On the reading side of the socket, we can simply go into an infinite loop and poll for data from the server, since the listener is running as a separate thread.

Java has most of the same basic data types as C, with a few exceptions. Java has no unsigned integers, but it does have true booleans. To construct the data packet, use a combination of Integers and an array of 120 bytes for the handle and message fields. The data output and input streams have methods for reading and writing integers and bytes. For example, **dos.writeInt(1);** would write the integer "1" to the data output stream. Conversely, **for (int i=0;i<120;i++) dos.writeByte(buffer[i]);** (*or* **dos.readByte(buffer[i])** to read) would write the entire buffer to the socket; **dos.flush()** will make sure that the data is written now and not delayed. It is important to note that we must write or read all of the data (command, channel, handle and message) to or from the server even if all we want to do is change the channel. The server expects this; otherwise it will hang, waiting for all of the bytes to come or go.

One more obstacle remains. How to get the handle and message data into the proper position in the byte array? In the event handler we create string objects for the message and handle, then call the **getBytes()** method on the string objects. **message.getBytes(0,message.length(),buffer,20);** will copy **message.length()** bytes from the string object **message** starting at position **0** in the string to the byte array **buffer** starting at position **20**. One thing that is missing in my program is error checking. It would be absolutely necessary in a production program to check and recheck to make sure that you don't overflow the buffer by writing more bytes than the buffer can hold.

The Server

The server is a simple single-process concurrent server. Simply put, the server polls each connection, and processes requests in order. An alternative would be to fork a new process for each incoming connection. In this situation the single process server is far simpler (and, after all, the computer can only really do one thing at a time). The basic order of things is:

1. Create the master socket on the well-known port.
2. Bind the socket so that incoming requests are directed to the proper place.
3. Listen for connections.
4. Accept incoming connections.

The “well-known port” is a port which is known to all clients. All clients can't connect to the same port, so the server “hands off” connection requests to a different port. This is done by the TCP/IP layer, and we don't need to concern ourselves about how. This process is analogous to that of making a phone call to a large corporation's toll free number. Suppose that you wish to call 1-800-257-1234. You are asking the server for a connection on that port (phone) number. This company probably has hundreds of lines, but you would not want to try each of them until you finally got through, so the corporation has set up a rotary on their lines to put connections through to the next available phone line.

TCP/IP sockets work the same way. When a connection is accepted on a socket, a new file descriptor is created. The file descriptor is used as an index to an array of structures. Every client has exactly one unique file descriptor and a slot in the client array. Each array position holds a structure which contains the handle and current channel number. When the server receives a message packet, it looks through the entire array and retransmits the message to all clients who are subscribed to the channel number that the message came from.

Currently supported server commands are:

- **CB_ON** sets the client's channel to 19 and sends a welcome message. It also stores the handle in the client array.
- **CB_OFF** closes the socket and clears the client info from the client array.
- **SET_CHAN** changes the channel of the client in the client array.
- **WHO_CHAN** sends a message containing the handles of all connected clients subscribed to the current channel.
- **SEND-MESSAGE** sends the message contained in the message field of the data packet to each client subscribed to the same channel as the

originator. Emergency traffic on channel 9 is also sent to all connected clients regardless of what channel they are subscribed to. As stated earlier, the server must have all bytes in the data packet sent or received at once. It is not possible (with this implementation) to send part of a packet.

The Client

My original goal was to make a client that looked like a CB radio panel. This turned out to be too difficult to do with Java; while Java is a good portable programming language, creating a complex user interface is very difficult. I adapted my client from an example in *Java in a Nutshell* by David Flanagan, O'Reilly & Associates (an excellent book—great for reference). The CB client user interface is very simple. A Connect menu is at the top. From here, the user can quit or ask the server who is on the current channel. The middle window is the message area. Here all messages from other users and the server are listed. The client will print the handle and message from the data packet. The server is responsible for the data in those packets as it will put “System: WHO” in the handle for a WHO request. The bottom field is for entering a new channel. When Java detects activity in the menu bar or channel field, it will call the event handler routine. From here, it determines where the event came from and performs the appropriate processing. The user interface is not much —more a “proof of concept” than anything else—but it does provide much more functionality in fewer lines of code than would be required by an equivalent program written in C.

Endian Wars

The big vs. little endian debate has been the topic of many flame wars on the Internet. But what is it? Big and little endian refers to the order of bytes. When moving data around, some systems start with the most significant byte (MSB) and some start with the least significant byte (LSB). Imagine an array of 4 bytes. How do you store or send this array? Would you start at the LSB (little endian) or would you start at the MSB (big endian)? Some hardware does it one way, and some does it the other. Why do we care? If you are writing a client and server in C to run on the same type of hardware, the endian problem doesn't pop up. But if you are using a different language, like Java, to talk to a server written in C, there could be a big problem. Endian problems crop up only when multiple byte data types like integers are sent across the network. Java automatically converts its data to and from network byte order when it sends data through a socket. C, on the other hand, does only as it is told. There are two C system calls, **ntohl()** and **htonl()**, which convert data to and from network byte order. Read the man pages for these calls and use them in your C-based servers and clients to avoid endian problems.

Java and Security

Java has some strict security restrictions. An applet can only open a socket to the server on which it was loaded. Applications, on the other hand, are allowed to open sockets to any machine. My client is written as a stand-alone application for this reason. (I don't have access to a web server that will allow me to run my CB server.) There are very few major differences between an **applet** and an application. An applet extends the class applet and an application extends the class **frame**. Refer to a book on Java for more specific details.

Conclusion

This project was my first real attempt at client-server programming. I'm hooked! With the basic server written, it is possible to extend the code to do many things. I would like to eventually redesign the user interface to make it look better and be easier to use. Having Linux at home has made the program development process much easier. I was able to use the same tools on both my home system and the Sun workstations at school so a simple recompilation was all that was necessary for the server to run on a Sparc 5. My hope is that someone else will find this work useful. No references could be found in any Java book (I have three) to address this specific application. While client-server applications were available in all of these books, all of the servers were written in Java. Java works well for writing servers, but is not as fast and requires more system resources to run. Every language has its place and Java is no exception. Java is very useful as a client programming language; it's here to stay.

Take a look at the listings:

- Listing 1
- Listing 2
- Listing 3

References

- *Java in a Nutshell*
- David Flanagan (1996, O'Reilly & Associates, Inc)
- *Java Programming Explorer*
- Neil Bartlett, Alex Leslie, and Steve Simkin (1996, Coriolis Group Books)
- *Teach Yourself Java in 21 Days*
- Lemay and Perkins (1996 Sams.net publishing)
- *Internetworking with TCP/IP vol III*
- Comer and Stevens (1993, Prentice-Hall, Inc)

- *The C Programming Language, second edition*
- Kernighan and Ritchie (1988, 1978, Prentice Hall)
- Various Linux man pages

Joe Novosel (jnovosel@cc.gatech.edu) has been an avid computer hobbyist since 1981, when his first computer (Radio Shack Color Computer) had a whopping 4K of memory (including video memory!). He has been using Linux for about two years—since version 1.1.47—and thinks Linux brings back the excitement of his early days in computing. After several years in the electrical trade, Joe decided to return to school and is now a Junior at Georgia Tech, where he pursues a degree in Computer Science.

Archive Index Issue Table of Contents

Advanced search

CGI Programming

Reuven M. Lerner

Issue #33, January 1997

So you're gathering information from your surfers; what now?

This time, we are going to look at one of the most common things that people want their CGI programs to do, namely save data to files on disk. By the end of the column, we will have accumulated enough tools to produce a simple, but functional guest-book program that will allow visitors to your site to save comments that can be read by others.

For starters, let's look at a simple HTML form that will allow users to send data to our CGI program, which we will call “entryform.pl”:

```
<HTML>
<Head>
<Title>Data entry form</Title>
</Head>
<Body>
<H1>Data entry form</H1>
<Form action="/cgi-bin/entryform.pl"
method=POST>
<P>Name: <input type=text name="name"
value=""></P>
<P>E-mail address: <input type=text
name="email" value=""></P>
<P>Address: <input type=text
name="address" value=""></P>
<P>Country: <input type=text
name="country" value=""></P>
<P>Male <input type=radio name="sex"
value="male">
Female <input type=radio name="sex"
value="female"></P>
<input type=submit>
</Form>
</Body>
</HTML>
```

Of course, an HTML form won't do anything on its own; it needs a CGI program to accept and process its input. Below is a Perl5 program that, if named “entryform.pl” and placed in the main “/cgi-bin” directory on a web server, should print out the name-value pairs that arrive from the above form:

```
0    #!/usr/local/bin/perl5
1    # We want to use the CGI module
2    use CGI;
3    # Create a new CGI object
4    my $query = new CGI;
5    # Print an a appropriate MIME header
6    print $query->header("text/html");
7    # Print a title for the page
8    print $query->start_html(-title=>"Form
     contents");
9    # Print all of the name-value pairs
10   print $query->dump();
11   # Finish up the HTML
12   print $query->end_html;
```

Here's a quick run-down of what each line of code does:

Line 0 tells a Unix box where to find the Perl interpreter. If your copy of Perl is called something else, you need to modify this line.

Without explicitly importing the CGI module in line 2, Perl wouldn't know how to create and use CGI objects. (Trying to use code from a module you haven't imported is guaranteed to confuse Perl and generate error messages.) We then declare $query to be an instance of CGI (line 4).

We then tell the user's browser that our response will be HTML-formatted text, and we do that by using a MIME header. The lack of a MIME header is the most common reason for a 500 error; whenever one of your CGI programs produces one of these, make sure that you aren't trying to print HTML before the header! Note that line 6 is equivalent to saying:

```
print "Content-type: text/html\n\n";
```

which also tells the browser to expect text data formatted in HTML. In general, though, I prefer to use the CGI object for readability reasons.

Line 8 creates the basic HTML necessary to begin the document, including giving it the title, “Form contents”.

Line 10 uses the CGI object's built-in facility for “dumping” an HTML form's contents in an easy-to-read format. This allows us to see what value was assigned to each of the elements of the HTML form, which can be invaluable in debugging problematic programs. For now, though, we are just using the CGI “dump” method to get ourselves started and confirm that the program works.

Saving the Data to a File

Now that we have proven that our HTML form is sending data to our CGI program, and that our program can send its output back to the user's web browser, let's see what we can do with that data. For starters, let's try to save the data from the form to a file on disk. (This is one of the most common tasks

that clients ask me to implement, usually because they want to collect data about their visitors.)

```
#!/usr/local/bin/perl5
# We want to use the CGI module
use CGI;
# Set the filename to which we want the elements
# saved
my $filename = "/tmp/formcontents";
# Set the character that will separate fields in
# the file
my $separation_character = "\t";
# Create a new CGI object
my $query = new CGI;
# ---------------------------------------------
# Open the file for appending
open (FILE, ">>$filename") ||
        die "Cannot open \"$filename\"!\n";
# Grab the elements of the HTML form
@names = $query->param;
# Iterate through each element from the form,
# writing each element to $filename. Separate
# elements with $separation_character defined
# above.
foreach $index (0 .. $#names)
{
        # Get the input from the appropriate
        # HTML form element
        $input = $query->param($names[$index]);
        # Remove any instances of
        # $separation_character
        $input =~ s/$separation_character//g;
        # Now add the input to the file
        print FILE $input;
        # Don't print the separation character
        # after the final element

print FILE $separation_character if
                ($index < $#names);
}
# Print a newline after this user's entry
print FILE "\n";
# Close the file
close (FILE);
# ------------------------------------------------
# Now thank the user for submitting his
# information
# Print an a appropriate MIME header
print $query->header("text/html");
# Print a title for the page
print $query->start_html(-title=>"Thank you");
# Print all of the name-value pairs
print "<P>Thank you for submitting the ";
print "form.</P>\n";
print "<P>Your information has been ";
print "saved to disk.</P7gt;\n";
# Finish up the HTML
print $query->end_html;
```

The above program is virtually identical to the previous one, except that we have added a section that takes each of the HTML form elements and saves them to a file. Each line in the resulting file corresponds to a single press of the HTML form's “submit” button.

The above program separates fields with a TAB character, but we could just as easily have used commas, tildes or the letter “a”. Remember, though, that someone is eventually going to want to use this data—either by importing it into a database or by splitting it apart with Perl or another programming

language. To ensure that the user doesn't mess up our database format, we remove any instances of the separation character in the user's input with Perl's substitution(s) operator. A bit Draconian, but effective!

One of the biggest problems with the above program is that it depends on the HTML form elements always coming in the same order. That is, if you have elements X, Y and Z on an HTML form, will they be placed in @names in the same order as they appear in the form? In alphabetical order? In random order? To be honest, there isn't any way to be sure, since the CGI specifications are silent on the matter. It's possible, then, that one user's form will be submitted in the order (X, Y, Z), while another's will be submitted as (Y, Z, X)—which could cause problems with our data file, in which fields are identified by their position.

A simple fix is to maintain a list of the fields that we expect to receive from the HTML form. This requires a bit more coordination between the program and the form, but given that the same person often works on both, that's a minor concern.

First, we define a list, **@fields**, near the top of the program. This list contains the names of all of the fields that we expect to receive, in the order that we expect to receive them:

```
my @fields = ("name",
                      "email",
                      "address",
                      "country",
                      "sex");
```

Next, we change the “foreach” loop (which places the field elements in the output file) such that it iterates through the elements of **@fields**, rather than **@names**.

```
foreach $index (0 .. $#fields)
{
  # Get the input from the appropriate HTML form
  # element
  $input = $query->param($fields[$index]);
  # Remove any instances of $separation_character

  $input =~ s/$separation_character//g;
  # Now add the input to the file
  print FILE $input;
  # Don't print the separation character after the
  # final element
  print FILE $separation_character if
        ($index < $#fields);
}
```

Required Fields

What if we want to make sure that users have filled out certain fields? This is particularly important when we are collecting data about visitors to a site, and

want to make sure that we receive their names, addresses and other vital data. A simple way to do that is to create a list, **@required_fields**, in which the required fields are listed:

```
my @required_fields = ("name",
                       "email",
                       "address");
```

If you simply want a generic message indicating that one or more required fields haven't been filled out, you can add the following subroutine at the bottom of the program file:

```
sub missing_field
{
  # Print an a appropriate MIME header
  print $query->header("text/html");
  # Print a title for the page
  print $query->start_html(-title=>
  "Missing field(s)");
  # Tell the user what the error is
  print "<P>At least one required ";
  print "field is missing.</P>\n";
  # Finish up the HTML
  print $query->end_html;

}
```

We can then insert the following code into the program itself, just before we open the file—since there isn't any reason to open the file if we are simply going to close it again:

```
foreach $field (@required_fields)
{
  # Make sure that the field contains more than
  # just whitespace
  &missing_field if
  ($query->param($field) !~m/\w/);
  exit;
}
```

The above code will indeed do the trick, but gives a generic error message. Wouldn't it be better to tell the user *which* field contains the error? We can do that by modifying missing_field such that it takes an argument, as follows:

```
sub missing_field
{
  # Get our local variables
  my (@missing_fields) = @_;
  # Print an a appropriate MIME header
  print $query->header("text/html");
  # Print a title for the page
  print $query->start_html
  (-title=>"Missing field(s)");
  print "<P>You are missing the following ";
  print "required fields:</P>\n";
  print "<ul>\n";
  # Iterate through the missing fields, printing
  # them foreach $field (@missing_fields)
  {
    print "<li> $field\n";
  }
  print "</ul>\n";
```

```
  # Finish up the HTML
  print $query->end_html;
  exit;
}
```

We then modify the loop that checks for required fields:

```
foreach $field (@required_fields)
{
  # Add the name of each missing field
  push (@missing_fields, $field) if
   ($query->param($field) !~ m/\w/);
}
# If @missing_fields contains any elements, then
# invoke the error routine
&missing_field(@missing_fields)
  if @missing_fields;
```

If we want to get really fancy, we can provide English names for each of the required fields, so that users don't have to suffer through the names we used with the HTML form. We can do that by using associative arrays:

```
$FULLNAME{"name"} = "your full name";
$FULLNAME{"email"} = "your e-mail address";
$FULLNAME{"address"} = "your mailing address";
```

Then we modify the foreach loop in &missing_fields such that it prints the full name of the missing field, rather than the name associated with it on the HTML form:

```
# Iterate through the missing fields, printing
# them foreach $field (@missing_fields)
{
  print "<li> $FULLNAME{$field}\n";
}
print "</ul>\n";
```

Dying with Style

Remember that **die** statement we put in our original program? Well, think about what will happen if that part of the program is ever truly invoked—**die** will produce an error message, which is a good thing. But that error message will be sent to our web browser, before the HTML header, giving us the dreaded "Server error" message, indicating that something (but not saying what that something is) has gone wrong with our script.

More useful would be a routine that printed the error message to the screen. For example, we could add the following subroutine:

```
sub error_opening_file
{
    my ($filename) = @_;
    # Print an a appropriate MIME header
    print $query->header("text/html");
    # Print a title for the page
    print $query->start_html(-title=>"Error
    opening file");
    # Print the error
    print "Could not open the file
```

```
    \"$filename\".</P>\n";
    # Finish up the HTML
    print $query->end_html;
    exit;
}
```

And now, we can rewrite the “open” statement as follows:

```
open (FILE, ">&gt:$filename") ||
  &error_opening_file($filename);
```

You probably don't want to tell your users your program couldn't open a particular file—not only do your users not care, but you don't need to tell them which files you are using. A more user-friendly version of **error_opening_file** could tell the user that the server is experiencing some trouble, or is undergoing maintenance or give a similar message that doesn't broadcast catastrophe to the world.

Bringing It All Together

The final version of the program, with (a) required fields, (b) full-English descriptions of those fields, and (c) a better error message when we cannot open the file, reads as follows:

```
#!/usr/local/bin/perl5
# We want to use the CGI module
use CGI;
# Set the filename to which we want the elements
# saved
my $filename = "/tmp/formcontents";
# Set the character that will separate fields in
# the file
my $separation_character = "\t";
# In what order do we want to print fields?
my @fields = ("name",
              "email",
              "address",
              "country",
              "sex");
# Which fields are required?
my @required_fields = ("name",
                       "email",
                       "address");
# What is the full name for each required field?
$FULLNAME{"name"} = "your full name";
$FULLNAME{"email"} = "your e-mail address";
$FULLNAME{"address"} = "your mailing address";
# Create a new CGI object
my $query = new CGI;
# --------------------------------------------
# Make sure that all required fields have arrived
foreach $field (@required_fields)
{
    # Add the name of each missing field
    push (@missing_fields, $field)
    if ($query->param($field) !~ m/\w/);
}
# If any fields are missing, invoke the error
# routine
&missing_field(@missing_fields)
  if @missing_fields;
# --------------------------------------------
# Open the file for appending
  open (FILE, "7gt;>$filename") ||
  &error_opening_file($filename);
# Grab the elements of the HTML form
```

```
@names = $query->param;
# Iterate through each element from the form,
# writing each element to $filename. Separate
# elements with $separation_character defined
# above.
foreach $index (0 .. $#fields)
{
    # Get the input from the appropriate HTML
    # form element
    $input = $query->param($fields[$index]);
    # Remove any instances of
    # $separation_character
    $input =~ s/$separation_character//g;
    # Now add the input to the file
    print FILE $input;
    # Don't print the separation character after
    # the final element
    print FILE $separation_character if
    ($index < $#fields);
}
# Print a newline after this user's entry
print FILE "\n";
# Close the file
close (FILE);
# ---------------------------------------------
# Now thank the user for submitting their
# information
# Print an a appropriate MIME header
print $query->header("text/html");
# Print a title for the page
print $query->start_html(-title=>"Thank you");
# Print all of the name-value pairs
print "<P>Thank you for submitting ";
print "the form.</P>\n";
print "<P>Your information has been ";
print "saved to disk.</P>\n";
# Finish up the HTML
print $query->end_html;
# ---------------------------------------------
# Subroutines
sub missing_field
{
    # Get our local variables
    my (@missing_fields) = @_;
    # Print an a appropriate MIME header
    print $query->header("text/html");
    # Print a title for the page
    print $query->start_html(-title=>
    "Missing field(s)");
    print "<P>You are missing the following
    required fields:</P>\0";
    print "<ul>\n";
    # Iterate through the missing fields,
    # printing them
    foreach $field (@missing_fields)
    {
        print "<li> $FULLNAME{$field}\n";
    }

    print "</ul>\n";

    print "</ul>\n";

    # Finish up the HTML
    print $query->end_html;

    exit;
}
sub error_opening_file
    {
        my ($filename) = @_;
        # Print an a appropriate MIME header
        print $query->header("text/html");
        # Print a title for the page
        print $query->start_html(-title=>
        "Error opening file");
        # Print the error
        print "Could not open the file
```

```
        \"$filename\".</P>\n";
        # Finish up the HTML
        print $query->end_html;
        exit;
    }
```

Creating a Guest-book

One of the most common CGI applications on the Web is a “guest-book”, which allows visitors to a site to sign in, leaving their names, e-mail addresses and short notes. We can easily construct such a program, using the basic framework seen in the above programs. The only difference between the “guestbook” program and the programs we have seen so far is that the guest-book must be formatted in HTML in order for users to be able to read it in their browsers.

Here is a very simple guest-book program that is virtually the same as the previous program we saw:

```
<HTML>
<Head>
<Title>Guestbook entry</Title>
</Head>
<Body>
<H1>Guestbook entry</H1>
<Form action="/cgi-bin/guestbook.pl"
    method=POST>
<P>Name: <input type=text name="name"
    value=""></P>
<P>E-mail address: <input type=text name="email"
value=""></P>
<input type=submit>
</Form>
</Body>
</HTML>
```

The following program is the same as the one above, except that it saves data to the “guestbook.html” and formats the data in HTML.

```
#!/usr/local/bin/perl5
# We want to use the CGI module
use CGI;
# Set the filename to which we want the elements
# saved
my $filename =
"/home/reuven/Consulting/guestbook.html";
# Set the character that will separate fields in
# the file
my $separation_character = "</P><P>";
# In what order do we want to print fields?
my @fields = ("name", "email");
# Which fields are required?
my @required_fields = ("name", "email");
# What is the full name for each required
# field?
$FULLNAME{"name"} = "your full name";
$FULLNAME{"email"} = "your e-mail address";
# Create a new CGI object
my $query = new CGI;
# --------------------------------------------
# Make sure that all required fields have arrived
foreach $field (@required_fields)
{
  # Add the name of each missing field
  push (@missing_fields, $field) if
```

```
  ($query->param($field) !~ m/\w/);
}
# If any fields are missing, invoke the error
# routine
&missing_field(@missing_fields) if
  @missing_fields;
# --------------------------------------------
# Open the file for appending
open (FILE, ">>$filename") ||
  &error_opening_file($filename);
# Grab the elements of the HTML form
@names = $query->param;
# Iterate through each element from the form,
# writing each element to $filename. Separate
# elements with $separation_character defined
# above.
foreach $index (0 .. $#fields)
{
  # Get the input from the appropriate HTML form
  # element
  $input = $query->param($fields[$index]);
  # Remove any instances of $separation_character
  $input =~ s/$separation_character//g;
  # Now add the input to the file
  print FILE $input;
  # Don't print the separation character after the
  # final element
  print FILE $separation_character if
  ($index < $#fields);
}
# Print a newline after this user's entry
print FILE "<BR><HR><P>\n\n";
# Close the file
close (FILE);
# ------------------------------------------
# Now thank the user for submitting his
# information
# Print an a appropriate MIME header
print $query->header("text/html");
# Print a title for the page
print $query->start_html(-title=>"Thank you");
# Print all of the name-value pairs
print "<P>Thank you for submitting ";
print "the form.</P>\n";
print "<P>Your information has been ";
print "saved to disk.</P>\n";
# Finish up the HTML
print $query->end_html;
# ------------------------------------------
# Subroutines
sub missing_field
{
  # Get our local variables
  my (@missing_fields) = @_;
  # Print an a appropriate MIME header
  print $query->header("text/html");
  # Print a title for the page
  print $query->start_html(-title=>"
  Missing field(s)");
  print "<P>You are missing the ";
  print "following required fields:</P>\n";
  print "<ul>\n";
  # Iterate through the missing fields, printing
  # them
  foreach $field (@missing_fields)
    {
        print "<li> $FULLNAME{$field}\n";
        }
  print "</ul>\n";
  print "</ul>\n";
  # Finish up the HTML
  print $query->end_html;
  exit;
}
sub error_opening_file
{
  my ($filename) = @_;
  # Print an a appropriate MIME header
```

```
    print $query->header("text/html");
    # Print a title for the page
    print $query->start_html(-title=>
    "Error opening file");
    # Print the error
    print "Could not open the ";
    print "file \"$filename\".</P>\n";
    # Finish up the HTML
    print $query->end_html;
    exit;
  }
```

The above program will take input from the HTML form and save the data in an HTML-formatted file. If that file is accessible from the web server, your users should be able to view others' entries in the guest-book.

Reuven M. Lerner (reuven@the-tech.mit.edu) (reuven@netvision.net.il) has been playing with the Web since early 1993, when it seemed like more of a fun toy than the world's Next Great Medium. He currently works from his apartment in Haifa, Israel as an independent Internet and Web consultant. When not working on the Web or informally volunteering with school-age children, he enjoys reading (just about any subject, but especially computers, politics, and philosophy—separately and together), cooking, solving crossword puzzles and hiking.

Archive Index Issue Table of Contents

Advanced search

unzip

Greg Roelofs

Issue #33, January 1997

Primer on zip, unzip, pkzip.

As much as we all love Linux, it is nevertheless true that occasionally we must force ourselves to deal with the DOS/MS-Windows world, however indirectly. For some of us that involves having a dual-boot system (perhaps via LILO—the LInux LOader—or OS/2's Boot Manager), but even those of us who manage to avoid that fate will sooner or later come across files that originated on some flavor of DOS or Windows system. More than likely, a few of those files will end in **.zip**—and that's where the **unzip** command comes in.

unzip is a free utility to process *zipfiles*, as these things are generally called. Zipfiles are actually archives of one or more other files, almost always compressed to save disk space and/or transmission time. In this regard they are similar to compressed **tar** archives, which are those files usually ending in **.tar.Z**, **.tar.gz** or **.tgz** that one finds on most Linux ftp sites and many CD-ROM distributions. One major difference between zip files and tar archives: compressed tar archives bundle all of the files together and then compress the result as a single entity; zipfiles compress individual files, then store them in the archive. This zip file method isn't quite as efficient in achieving the maximal overall compression, but it does allow you to list the archive's contents and to extract individual files without decompressing the whole mess.

Listing

How does one actually use unzip to list an archive's contents? The simplest way is with the **-l** option (for “list”):

```
$ unzip -l quake92p.zip
Archive:  quake92p.zip
 Length     Date    Time    Name
 ------     ----    ----    ----
  36064  06-25-96  13:18   DEICE.EXE
 369135  06-27-96  03:51   QUAKE92P.1
   2618  06-27-96  03:34   README.TXT
```

```
      177  06-25-96  20:07   INSTALL.BAT
      206  06-27-96  03:54   QUAKE92P.DAT
   ------                    -------
   408200                    5 files
```

You have each file's name (on the right), its uncompressed size, and the date and time of its last modification. For many of us, however, especially those long steeped in the terse intricacies of **ls**, this is a little *too* short and sweet. For fans of ls, or for anyone wishing to know more about the details of the archive, unzip has an entire mode devoted to listing both useful and obscure zipfile information: zipinfo mode, triggered via the **-Z** option. (On some systems the **zipinfo** command exists as a link to **unzip** and is synonymous with **unzip -Z**, but this is not true of Slackware distributions as of this writing.) We'll limit ourselves to a description of the default zipinfo listing format:

```
$ unzip -Z quake92p.zip
Archive:  quake92p.zip   406075 bytes   5 files
-rwxa--     2.0 fat   36064 b- defN 25-Jun-96 13:18 DEICE.EXE
-rw-a--     2.0 fat  369135 b- stor 27-Jun-96 03:51 QUAKE92P.1
-rw-a--     2.0 fat    2618 t- defN 27-Jun-96 03:34 README.TXT
-rwxa--     2.0 fat     177 t- defN 25-Jun-96 20:07 INSTALL.BAT
-rw-a--     2.0 fat     206 t- defN 27-Jun-96 03:54 QUAKE92P.DAT
5 files, 408200 bytes uncompressed, 405569 bytes compressed:  0.6%
```

You will immediately recognize a certain resemblance to the output of **ls -l**. The header line gives the archive name, its total size, and the total number of files in it; the trailer gives the number of files listed (in this case all of them), the total uncompressed and compressed data size of the listed files (not counting internal zipfile headers), and the compression ratio. Here the ratio is quite poor, mostly due to the fact that the largest file (QUAKE92P.1) is stored without any compression. In the leftmost column are the file permissions. The next column indicates the version of the archiver, and the one after that is what tells us the files came from the FAT (DOS) file system. Next are the uncompressed file size and a column indicating which files are most likely to be **b**inary and which are probably **t**ext. The next three columns note the compression method used on each file; the time stamps; and the full file names.

Extracting

Now that we know what files we have, how do we actually get the files out? File extraction is as simple as typing **unzip** and the file name:

```
$unzip quake92p
Archive:  quake92p.zip
  inflating: DEICE.EXE
  extracting: QUAKE92P.1
  inflating: README.TXT
  inflating: INSTALL.BAT
  inflating: QUAKE92P.DAT
```

Here we've omitted the **.zip** suffix; unzip first looks for the file **quake92p** and, not finding it, checks for **quake92p.zip** instead. What if we wanted only the

README.TXT file? No problem. Anything (well, almost anything) after the zipfile name is taken to be the name of one of the enclosed files:

```
$unzip quake92p README.TXT
Archive:  quake92p.zip
 inflating: README.TXT
```

Here you may notice a little snag. If you now edit this file in Linux with an editor like **vi**, you'll see what looks like **^M** at the end of each and every line. Or, if you view the file with a pager like **more**, you'll discover that any line uncovered by the `--More--` prompt gets erased immediately. These problems are due to the fact that DOS and its successors store text files with *two* end-of-line characters, CR and LF (a.k.a. carriage return and linefeed, respectively, or ^M and ^J, or CTRL-M and CTRL-J), rather than the more efficient single character (LF) used on all Unix systems. So when a Unix utility—like an editor or a pager or a compiler—looks at a DOS text file, it may behave a little oddly or die altogether.

Fortunately there's a simple solution: unzip's **-a** option. Originally a mnemonic for *ASCII conversion*, the option these days is used for all sorts of text-file conversions. As a single-letter option it does its best to automatically convert files that are supposedly text, while leaving alone those that are marked binary. **Be careful!** zip and PKZIP don't always guess correctly when creating the archive, particularly for certain classes of MS-Windows files, and unzip's "text" conversions are *almost always irreversible*. In other words, don't extract with auto-conversion and then delete the original zipfile without first making sure everything is Okay. unzip does indicate which files it thinks are text when auto-converting, however:

```
$ unzip -a quake92p
Archive:    quake92p.zip
inflating:  DEICE.EXE               [binary]
extracting: QUAKE92P.1              [binary]
inflating:  README.TXT              [text]
inflating:  INSTALL.BAT             [text]
inflating:  QUAKE92P.DAT            [text]
```

In this case everything worked as intended. If, for some reason, zip marked a text file as binary and you want to force text conversion, simply double the option: **-aa**.

But wait, there's more! The discriminating Linux user, happily accustomed to a file system that not only preserves the case of file names but also distinguishes between names differing only in case, is not going to settle for a bunch of all uppercase DOS file names in his or her directories. Enter the **-L** option. If (and only if) the file came from a single case file system like DOS FAT or VMS, **unzip -L** will convert it to lowercase upon extraction, thusly:

```
$ unzip -aL quake92p
Archive:  quake92p.zip
  inflating: deice.exe               [binary]
  extracting: quake92p.1             [binary]
  inflating: readme.txt              [text]
  inflating: install.bat             [text]
  inflating: quake92p.dat            [text]
```

Isn't that nice?

Testing

So now you've just downloaded a whole bunch of zipfiles but don't want to unpack them just to make sure they're Okay. What's the solution? Use the **-t** option to test them:

```
$ unzip -t quake92p
Archive:  quake92p.zip
    testing: DEICE.EXE                OK
    testing: QUAKE92P.1               OK
    testing: README.TXT               OK
    testing: INSTALL.BAT              OK
    testing: QUAKE92P.DAT             OK
No errors detected in compressed data of quake92p.zip.
```

Here we tested only one, and the output is a little too verbose—we really want only the one-line summary for each archive. unzip supports both a **-q** option for various levels of quietness (the more q's, the quieter) and the concept of wildcards, both for the internal files and for the zipfiles themselves:

```
$ unzip -tq \*.zip
No errors detected in compressed data of arena2b-grr.zip.
No errors detected in compressed data of PngSuite.zip.
No errors detected in compressed data of libgr2-elf-install.zip.
No errors detected in compressed data of ppmz-7.3.zip.
arithc.c               bad CRC e220fe9c  (should be 1c24998c)
At least one error was detected in macm.zip.
No errors detected in compressed data of xfer-zip151.zip.
No errors detected in compressed data of quake091.zip.
No errors detected in compressed data of quake92p.zip.
No errors detected in compressed data of p93b2200.zip.
```

```
8 archives were successfully processed.
1 archive had fatal errors.
```

Note that the wildcard character (“*”) is escaped with a backslash (“\”). Most shells expand wildcards themselves, and if we allowed that, unzip would see the command line as a list of archives; it would treat the first one as the zipfile name and the rest as files to be tested within the first one. By escaping the wildcard, we allow unzip to do its own directory search and wildcard-matching —which, incidentally, has the advantage that Unix-style *regular expressions* (very powerful wildcards) can be used not only under Linux but under all of the operating systems for which unzip ports exist, even plain old DOS.

The other thing to notice is that one of the archives has an error in it. Perhaps there was a transmission error, or maybe the original was damaged when it

was created; either way, the file **arithc.c** in **macm.zip** is probably not going to be usable. It's always good to know these sorts of things sooner rather than later.

There are quite a few other options and modifiers not covered here; a full tutorial would occupy most of this magazine. Fortunately, the unzip and zipinfo man pages (**man unzip** and **man zipinfo**) contain a complete listing of all of the options and examples for many of them. Unfortunately, Slackware 3.0 and earlier don't include the zipinfo man page. An abbreviated summary of zipinfo's options is available by typing **unzip -Z** . Similarly, a summary of most of unzip's options can be had simply by typing **unzip** with no parameters.

History, Acknowledgments, Pointers and the Future

unzip, zipinfo, zip and their kin were written by the Info-ZIP group, an Internet-based collection of strange beings from another universe who are currently scattered all over the planet. Yours truly (that would be me) is the principal author of unzip and zipinfo, but literally hundreds of people have contributed to them. Originally based on code by Samuel H. Smith, unzip has since been completely rewritten, with the exception of one routine which is no longer included by default. Nevertheless, we certainly owe him a debt of gratitude for getting us into this pickle. It would probably also be nice to mention the folks at PKWARE, whose PKZIP and PKUNZIP programs are the source of most of the DOS-originated zipfiles in the world. Note that Info-ZIP's programs are intended to be compatible with PKWARE's zipfiles, but they are not clones of PKWARE's programs. (For example, unzip recreates stored zipfile directory trees by default, whereas PKUNZIP requires a special option to do it.

Note also that while zip and **gzip** (sometimes called “GNU zip”) have similar names, a similar heritage—Jean-loup Gailly and Mark Adler are the co-authors of the latter and are also long-standing members of the Info-ZIP group—and the same compression engine, the two programs are basically incompatible. The same goes for unzip and **gunzip**. Jean-loup never foresaw the confusion that would arise from the similarity, and I was too late in suggesting the obvious, sick alternative (**feather***) to get the name changed.

On a more serious note, the current version of unzip is 5.2, and 5.21 will be out by the time you read this. While everything discussed above works equally well with the previous version (5.12), there are various new features and other improvements that make 5.2 worth getting. You can find the latest public releases of source code and executables at UUNET's anonymous ftp site:

ftp://ftp.uu.net/pub/archiving/zip/ ftp://ftp.uu.net/pub/archiving/zip/UNIX/LINUX/

You can also find news, history, descriptions of certain weirdos, and pointers to other ftp sites around the world at the following web site:

http://quest.jpl.nasa.gov/Info-ZIP-

Greg Roelofs escaped from the University of Chicago with a degree in astrophysics and fled screaming to Silicon Valley, where he now does really cool graphics and compression stuff for Philips Research. He joined Info-ZIP in the spring of 1990, shortly after the group formed, and under his dark influence the group has nearly achieved its goal of Total Universal Reconstructive Disintegration, lacking only a better acronym in order to complete their plans. He's also the father of the Cutest Baby in the Known Universe. He be reached by e-mail at newt@uchicago.edu or on the Web at quest.jpl.nasa.gov/Info-ZIP/people/greg/.

[---- this is a footnote ----] * So all of the archives on Sunsite would be...yes, you guessed it: **tar**'d and **feather**'d. Bwah ha ha ha ha ha ha! If only.

Archive Index Issue Table of Contents

Advanced search

New Products

Margie Richardson

Issue #33, January 1997

High Speed Intelligent Multi-port Card, TEAMate Server Software, SPATCH Alphanumeric Paging Software Upgrades and more.

High Speed Intelligent Multi-port Card

Cyclades Corporation announced its next generation of high performance intelligent PCI multi-port card—CYCLOM-8Zo. This card has a 32-bit RISC processor, PCI bus and RS232 serial channels and built-in surge protection. CYCLOM-8Zo is able to sustain baud rates of 460Kbps full duplex simultaneously on all 8 channels with low CPU overhead. It is available with octopus cable for $695 (special promotional price of $421 to ISPs).

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

TEAMate Server Software

MMB Development announced TEAMate Server Software, a package of the 9 most popular out-of-the-box web applications available for Linux at a price of $895. The nine applications included are Chat, Forums, Classified Ads, Mail, Catalogs, Maps, Event Calendars, User Home Pages and Server Content Indexing. Included in the package are TEAMate's web, VT100 and GUI interfaces.

Contact: MMB Development Corporation, 904 Manhattan Ave., Manhattan Beach, CA 90266, Phone: 800-832-6022, Fax: 310-318-2162, E-mail: query@mmb.com, http://teamate.mmb.com.

SPATCH Alphanumeric Paging Software Upgrades

The Hyde Company has announced several upgrades to its base SPATCH alphanumeric paging software for Unix, includes Linux. With SPATCH users can send text messages to an alphanumeric pager, an application, an operating system or an e-mail system. One, Version 3 of it ISP solution to give ISPs the ability to offer an e-mail forwarding service to their clients has been released. Two, SPATCH-XL has been released for use by companies that need multiple modems to send alpha pages (supports 2 to 256 modems). Three, PagePage has been released to allow alphanumeric paging on an Intranet and works in any Intranet/Internet environment. Four, Echelon Page has been released to enhance the abilities of the SPATCH command line interface which allows users to send messages about system alarms and events to a pager. SPATCH base package starts at $199 for single user.

Contact: The Hyde Company, P.O. Box 900190, Atlanta, Georgia 30329, Phone: 770-495-0718, Fax: 770-476-7626, E-Mail: spatch@america.net, http://www.spatch.com/.

New Linux Books and Software

O'Reilly & Associates announces several new offerings for the Linux enthusiasts. *Linux Multimedia Guide* by Jeff Tranter is the first guide for developing multimedia applications using the Linux operating system. The second editions of *Running Linux* by Matt Welsh and Lar Kaufman and the *Running Linux Companion CD-ROM* (Red Hat Software) together provide everything one needs to install and use Linux on a PC. O'Reilly also added *Linux in a Nutshell* by Jessica Hekman to its Nutshell reference series. For pricing contact O'Reilly & Associates, Inc.

Contact: O'Reilly & associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 95472, Phone: 800-998-9938, Fax: 707-829-0104, E-mail: sara@ora.com.

Accelerated-X Server for Linux

WorkGroup Solutions, Inc. announced the release of Version 2.1 of Accelerated-X Server for Linux. Accelerated-X is the fastest, easiest to install and most compatible Xserver available. It contains a full Linux port of the X Windows X11R6.1 Graphics Engine, support more graphics boards than ever before, and is CDE-read (Common Desktop Environment) and capable. Accelerated-X Version 2.1 was developed by X Inside, and greatly improves graphics speed. It is available for $99.

Contact: WorkGroup Solutions, Inc., Department WEX016, P.O. Box 460190, Aurora CO 80046-0190, Phone: 800-234-7813, Fax: 303-699-2703, E-mail: sales@wgs.com, http://www.wgs.com/.

Red Hat Linux 4.0

Red Hat Software announced today the availability of Red Hat Linux 4.0. Red Hat Linux version 4.0 features many substantial improvements. These include more hardware support, simplified installation, dramatic performance improvements, and many more. The most important new feature is Red Hat Linux can now be used on Sun Microsystems SPARC and compatible computers, in addition to Digital Alpha, and Intel compatible PC platforms. Red Hat Linux 4.0 for Intel compatible computers is available for only $49.95. Red Hat Linux 4.0 for SPARC and Alpha editions are available immediately ready-to-install from CD-ROM for only $99.95 per copy.

Contact: Red Hat Software, Inc., 3203 Yorktown Rd, Suite 123, Durham, NC 27713, Phone: 800-454-5502, Fax: 919-572-6726, E-mail: bob@redhat.com, http://www.redhat.com/.

Silent Messenger for Linux

Silent Messenger from MessageNet Systems is a multiuser client/server paging gateway. It allows users to easily format messages and send that message via alphanumeric, digital and vibrating pagers. Software can be downloaded free with some restrictions (no support) from the web at http://www.trader.com/users/5013/3977/smparts.htm. Pricing depends on number of pagers, display boards and users—contact company directly.

Contact: MessageNet Systems, 4825 Pinebrook Dr., Novlesville, IN 46060, Phone: 800-577-2613, E-mail MessageNet@trader.com, http://www.trader.com/users/5013/3977/smparts.htm.

RAStel—integrated multi-link voice, data and fax card

Australian communications developer Moreton Bay Ventures has announced the release of RAStel, a remote-access solution that integrates multiple-link voice, data and fax in a single PC communications card. RAStel is available with four V.34+ modem ports or alternatively, two V.34+ modem ports and two high speed serial ports. Using multiple RAStel cards, a single PC server can connect directly to up to 32 telephone lines. RAStel supports all leading desktop and server operating systems including Linux. RAStel is immediately available from Moreton Bay Ventures directly or through its distribution partners in Australia and USA. The four modem RAStel model has a list price of $1495USD.

Moreton Bay Ventures, PO Box 925, Kenmore QLD 4069, Australia, Tel: +61 7 3279-1822, Fax: +61 7 3279-1820, Email: sales@moreton.com.au, http://www.moreton.com.au/moreton/

Mylex BusLogic SCSI Adapters for Linux

Mylex Corporation has expanded Linux operating system support to its BusLogic brand of FlashPoint Ultra SCSI host adapters. All of BusLogic's other SCSI host adapters, including the MultiMaster line, currently support the Linux operating system. Linux drivers and information are available at http://www.dandelion.com/Linux/.

Contact: Mylex Corporation, 34551 Ardenwood Blvd., Fremont, CA. 94555, Phone: 800-77-MYLEX, Fax: 510-745-7654, E-mail: peters@mylex.com, http://www.mylex.com/.

Archive Index Issue Table of Contents

Advanced search

Two Cent Tips

Marjorie Richardson

Issue #33, January 1997

Newbie Tip on Finding

As a new Linux user, I discovered that I had a hard time finding my way around the file system. I knew the **find** command was out there, but remembering all the options required to make it search the right places, find the right files, and print the right answers was something I couldn't do, at first. So I made up my own command, using a shell script, and called it **fnd**.

fnd takes one argument, the name of the file you want to locate, complete with any wildcards you may wish to include, and pipes its output to **less**, which then allows you to view a large list of results. What you get, on each line of output, is the complete path to any file that you're looking for. I find it amazingly useful (as is a rough familiarity with the **less** command). Here's my script:

```
!/bin/bash
find / -iname $1 -mount -print |less
```

That's it. The **-iname** option tells find to be case insensitive, the **$1** is a variable that substitutes in your command line argument, **-mount** tells **find** not to search directories on other file systems like your CD-ROM (because mine is wonky and locks up the machine if it is accessed). The **-print** option is required or you don't get any output. (Get used to it, it's *nix.) The | (pipe) symbol tells **find** to direct its output to the **less** command, so you can see your results in style. Don't forget the / right after the **find** command, or it won't know where to look. Enjoy. You won't regret the time you spend keying in this little shortcut, and don't forget to put it in a bin or sbin directory after using **chmod** to make it executable. —Jim Murphymurphyc@cadvision.com

X Term Titlebar Function

In the mail from issue 9, Jim Murphy says that the **-print** option to **find** is necessary to get output from the **find** command, and follows that up with “get

used to it, it's *nix." Well, he's part right. Linux does require this. However, any users who work on other Unix boxes will find slight differences in some of the common CLI (Command Line Interface) commands. For example, **find** on Solaris does not require the **-print** option to get output. Just food for thought.

Second, I have an xterm title bar function that people might find useful. I'll give the code first, then explain what it does.

In your .bashrc (or .kshrc—note this only works on ksh style shells) add the following:

```
HOSTNAME=`uname -n`
if [ "$TERM" = x"term" ] && [ "$0" =
"-bash"]
then<\n>
  ilabel () { echo -n "^[]1;$*^G"; }
  label () { echo -n ^"[]2;$*^G"; }
  alias stripe='label $HOSTNAME - ${PWD#$HOME/}'
  alias stripe2='label $HOSTNAME - vi $*'
  cds () { "cd" $*; eval stripe; }
  vis () { eval stripe2; "vi" $*; eval stripe;}
  alias cd=cds
  alias vi=vis
  eval stripe
  eval ilabel "$HOSTNAME"
fi
```

This does three things (as long as you're in an xterm and running bash):

1. When the xterm is first opened, the name of the current host is displayed in the title bar.
2. When you change directories (using cd), the current path is displayed in the xterm title bar with the user's $HOME directory stripped off the front end of the path (to save some space when you're somewhere in your own directory tree). The path is preceded by the current host's network name.
3. When you use vi to edit a file, the name of the file is displayed in the title bar along with the current host's name. When you exit your vi session, the title bar reverts to the hostname/path format described in #2 above.

I find this very useful for all my ksh-based systems, because it removed the path from my shell prompt, thus saving me space for prompt commands. Since bash is a ksh compatible shell, this works quite well on standard Linux systems. —Michael J. Hammelmjhammel@csn.net

Find and Alternatives

Saw Jim Murphy's **find** tip in issue #9, and thought you might like a quicker method. I don't know about other distributions, but Slackware and Red Hat come with the GNU versions of **locate(1)** and **updatedb(1)**, which use an index to find the files you want. The **updatedb(1)** program should be run once a night

from the crontab facility. To ignore certain sub-directories (like your /cdrom), use the following syntax for the crontab file:

```
41 5 * * * updatedb --prunepaths="/tmp /var \
        /proc /cdrom" > /dev/null 2>&1
```

This command would run every morning at 5:41 AM, and update the database with file names from everywhere except the subdirectories (and those below) listed.

To locate a file, just type **locate *file name***. The file name doesn't have to be complete; **locate** can also do partial matching. For me, the search typically takes only a few seconds, and I have tens of thousands of files.

The **locate(1)** command also has regular expression matching, but I often just pipe it through **agrep(1)** (a faster **grep**) to narrow the search. Thus:

```
locate locate | agrep -v man
```

would exclude the man page, and only show me the binary and the sources, if I had them on-line. (The **-v** flag excludes the pattern used as an argument.) To get the binary files alone, along with a complete directory listing, use the following command:

```
ls -l `locate locate | agrep bin`
```

The **find(1)** command is a great “swiss-army knife” (and actually not that bad once you get used to it), but for the 90% of the cases where you just want to search by file name, the **locate(1)** command is far faster, and much easier to use. —Bill Duncan, VE3IEDbduncan@ve3ied.uucp

Archive Index Issue Table of Contents

Advanced search