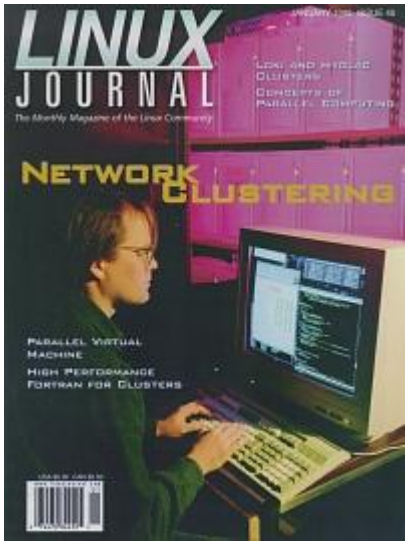
Linux Journal Issue #45/January 1998



Features

Parallel Computing Using Linux *by Manu Konchady*
Various classes of problems lend themselves to parallel computing solutions. This article discusses the concepts and shows how Linux can be used to address the problem.

Parallel Processing using PVM *by Richard A. Sevenich*
PVM is a software application that allows you to turn TCP/IP networked computers into a single virtual machine in order to run parallel programming.

I'm Not Going to Pay a Lot for This Supercomputer! *by Jim Hill, Michael Warren and Patrick Goda*
Los Alamos National Laboratory and Caltech obtain gigaflops performance on parallel Linux machines running free software and built of commodity parts costing less than $55,000 each.

HPF: Programming Linux Clusters the Easy Way *by Mike Delves*
Mr. Delves tells us all about high performance Fortran and how it is used to write code to run efficiently on parallel computers.

News & Articles

X-CD-Roast: CD Writer Software *by Thomas Niederreiter*
Mr. Niederreiter tells us all about his graphical user interface for writing data to a CD-ROM.

Netatalk, Linux and the Macintosh *by Richard Parry*
With Netatalk, you can drag and drop files from Linux to Mac and back, share system resources and more.

LJ Interviews Mike Apgar, Speakeasy Café *by Marjorie Richardson*
Interview
The Quick Start Guide to the GIMP, Part 3 *by Michael J. Hammel*
This month we learn how to use the Image Window and layers in building our images with the GIMP, a Linux power tool for the graphics artist.

Reviews

Ricochet Modem *by Randy Bentson*
Red Hat CDE *by Don Kuenz*
Microway “Screamer 533” *by Bradley Willson*
Running Linux *by Zach Beane*
JDBC Developer's Resource *by Rob Wehrli*
Unix for the Hyper-Impatient *by Daniel Lazenby*

WWWsmith

Internet Connections With the 56Kbps Modems *by Tony Williamitis*
Higher speed Internet connections are on the horizon with U.S. Robotics' XS modem and Rockwell International's K56Plus.
At the Forge A Recipe for Making Cookies *by Reuven M. Lerner*
Cookies are an excellent way of keeping track of users who visit a web site. Here's how to use them.

Columns

Letters to the Editor
From the Editor The Beowulf Project *by Marjorie Richardson*
Stop the Presses LISA '97 Conference *by Phil Hughes*
Linux Apprentice Need More Info? *by Bill W. Cunningham*
Need More Info? Here's how to get the information you need using GNU's hypertext system called info.
Take Command Kill: The Command to End All Commands *by Dean Provins*
Kill: The Command to End All Commands Need to get rid of a job that's gotten into a loop and refuses to end? Here's a command that will take care of the problem.
Linux Means Business Linux at Rancho Santiago College *by Steve Moritsugu*
Linux at Rancho Santiago College Linux is being used to teach Computer Science classes at a community college in Santa Ana, California.
New Products
System Administration Securing Networked Applications with SESAME *by Paul Ashley and Bradley Broom*
Securing Networked Applications with SESAME This article describes the SESAME Security Architecture and how it can be used to secure your networked applications.
Kernel Korner Playing with Binary Formats *by Alessandro Rubini*
Playing with Binary Formats This article explains how kernel modules can add new binary formats to a system and show a pair of examples.

Extra: Geek Vocabulary This article explains some of the common Geek terms.

Linux Gazette Remote Compilation Using ssh and make *by John R. Daily*

Remote Compilation Using ssh and make Here's a quick lesson in setting up scripts to use the ssh and make commands for compiling on a remote machine.

Best of Technical Support

Archive Index

Advanced search

Parallel Computing Using Linux

Manu Konchady

Issue #45, January 1998

Various classes of problems lend themselves to parallel computing solutions. This article discusses the concepts and shows how Linux can be used to address the problem.

Parallel computing involves the design of a computing system that uses more than one processor to solve a single problem. For example, if two arrays with ten elements each must be added, two processors can be used to compute the results. One processor computes the sum of the first five elements and the second processor computes the sum of the second five elements. After the computation, the results from one processor must be communicated to the other processor. Before starting the computation, both processors agree to work on independent sub-problems. Each processor works on a sub-problem and communicates when the solution is available. Theoretically, a two-processor computer should add the array of numbers twice as fast as a single-processor computer. In practice, there is overhead and the benefits of using more processors decrease for larger processor configurations.

Affordable Supercomputers

Obtaining a Unix workstation for the cost of a PC has been one of the benefits of using Linux. This idea has been carried a step further by linking together a number of Linux PCs. Several research projects are underway to link PCs using high performance networks. High speed networking is a hot topic and there are a number of projects using Linux to develop a low latency and high bandwidth parallel machine. (One URL is http://yara.ecn.purdue.edu/~pplinux.)

Currently, there is not much high level support for shared memory programming under SMP Linux. The basic Linux mechanisms for sharing memory across processors are available. They include the System V Inter-Processor Communication system calls and a thread library. But, it will be some time before a parallel C or C++ compiler will be available for Linux. Parallel

programming can still be done on an SMP Linux machine or on a cluster of Linux PCs using message passing.

Parallel computing is advantageous in that it makes it possible to obtain the solution to a problem faster. Scientific applications are already using parallel computation as a method for solving problems. Parallel computers are routinely used in computationally intensive applications such as climate modeling, finite element analysis and digital signal processing. New commercial applications which process large amounts of data in sophisticated ways are driving the development of faster computers. These applications include video conferencing, data mining and advanced graphics. The integration of parallel computation, multimedia technology and high performance networking has led to the development of *video servers*. A video server must be capable of rapidly encoding and decoding megabytes of data while simultaneously handling hundreds of requests. While commercial parallel applications are gaining popularity, scientific applications will remain important users of parallel computers. Both application types are merging as scientific and engineering applications use large amounts of data and commercial applications perform more sophisticated operations.

Parallel computing is a broad topic and this article will focus on how Linux can be used to implement a parallel application. We will look at two models of parallel programming: message passing and shared memory constructs.

Message Passing

Conceptually, the idea behind message passing is simple—multiple processors of a parallel computer run the same or different programs, each with private data. Data is exchanged between processors when needed. A message is transmitted by a *sender* processor to a *receiver* processor. One processor can be either a sender or a receiver processor at any time. The sender processor can either wait for an acknowledgement after sending or it can continue execution. The receiver processor checks a message buffer to retrieve a message. If no message is present, the processor can continue execution and try again later or wait until a message is received. Multiple *sends* and *receives* can occur simultaneously in a parallel computer. All processors within the parallel computer must be inter-connected by a network (Figure 1).

Figure 1. A Parallel Computer with Distributed Memory

All processors can exchange data with all other processors. The routing of messages is handled by the operating system. The message-passing model can be used on a network of workstations or within a tightly coupled group of

processors with a distributed memory. The number of hops between processors can vary depending on the type of inter-connection network.

Message passing between processors is achieved by using a communication protocol. Depending on the communication protocol used, the send routine usually accepts a destination processor ID, a message type, the start address for the message buffer and the number of bytes to be transmitted. The receive routine can receive a message from any processor or from a particular processor. The message can be of any particular type.

Most communication protocols maintain the order in which messages are sent between a pair of processors. For example, if processor 0, sends a message of type *a* followed by a message of type *b* to processor 1, then when processor 1 issues a receive from processor 0 for a generic message type, the message of type *a* will be received first. However, in a multi-processor system, if a processor issues a receive from any processor, there is no guarantee of the order of messages received from the sending processors. The order in which messages are transported depends on the router and the traffic on the network. To maintain the order of messages sent, the safest way is to use the source processor number and message type.

Message passing has been used successfully to implement many parallel applications. But a disadvantage of message-passing is the added programming required. Adding message-passing code to a large program requires considerable time. A domain decomposition technique must be chosen. Data for the program must be divided such that there is minimal overlap between processors, the load across all processors is balanced and each processor can independently solve a sub-problem. For regular data structures, the domain decomposition is fairly straightforward, but for irregular grids, dividing the problem so that the load is balanced across all processors is not trivial.

Another disadvantage of message passing is the possibility of deadlock. It is very easy to hang a parallel computer by misplacing a call to the *send* or *receive* routines. So, while message passing is conceptually simple, it has not been adopted fully by the scientific or commercial communities.

Shared Memory Constructs

Another approach to parallel programming is the use of shared memory parallel language constructs. The idea behind this scheme is to identify the parallel and sequential regions of a program (Figure 2). The sequential region of a program is run on a single processor while the parallel region is executed on

multiple processors. A parallel region consists of multiple threads which can run concurrently.

Figure 2. Parallel and Sequential Regions of a Program

For some programs, identifying parallel and sequential regions may be a trivial task, but for other programs it may involve modifying the sequential program to create parallel regions. The easiest approach is to rely on the compiler to determine parallel regions. This is the *automatic parallelization* approach which usually gives poor results. Most compilers are conservative when introducing parallelism and will not parallelize code if there is any ambiguity. For example, if elements of an array **x** are accessed through an index array, e.g., **x(index(i))**, in a loop, the loop will not be parallelized since the compiler cannot know if the elements of array **x** will be accessed independently in the loop.

The time-consuming part of any program is usually spent in executing loops. Parallel regions are created for a loop to speed up the execution of a loop. For the compiler to build a parallel region from a loop, the private and shared data variables of the loop must be identified. The example below is for the Silicon Graphics Challenge machine, but the concepts are similar for other shared-memory machines such as the Digital Alpha, IBM PC or Cray J90. Shared-memory constructs are placed before and after the loop.

```
#pragma parallel
#pragma shared (a,b,c)
#pragma local (i)
#pragma pfor iterate (i=0;100;1)
 for (i = 0; i < 100; i++) {
  a(i) = b(i) * c(i)
 }
#pragma one processor
```

The code before the first pragma statement and after the last pragma statement is executed on a single processor. The code in the loop is executed in parallel using multiple threads. The number of threads used is based on an environment variable. Identifying shared and private variables is easy for simple loops, but for a complex loop with function calls and dependencies it can be a tedious job.

Programming using shared-memory constructs can be simpler than message passing. A parallel version of a program using shared-memory constructs can be developed more rapidly than a message-passing version. While the gain in productivity may be appealing, the increase in performance depends on the sophistication of the compiler. A shared-memory compiler for parallel programs must generate code for thread creation, thread synchronization and access to shared data. In comparison, the compiler for message-passing is simpler. It consists of the base compiler with a communication library. While no one can predict better performance using shared-memory constructs or

message-passing, the message-passing model offers more scope for optimization. Data collection and distribution algorithms can be optimized for a particular application. Domain decomposition can be performed based on communication patterns in the code.

Another advantage of message passing is portability. A message-passing program written using a standard communication protocol can be ported from a network of PCs to a massively parallel supercomputer without major changes. Message passing can also be used on a network of heterogenous machines.

Each vendor of a shared memory compiler has chosen a different syntax for compiler directives. Therefore, code parallelized using directives is usually restricted to a particular compiler.

What is PVM?

Parallel Virtual Machine (PVM) is currently the most popular communication protocol for implementing distributed and parallel applications. It was initially released in 1991 by the Oak Ridge National Laboratory and the University of Tennessee and is being used in computational applications around the world. PVM is an on-going project and is freely distributed. The PVM software provides a framework to develop parallel programs on a network of heterogenous machines. PVM handles message routing, data conversion and task scheduling. An application is written as a collection of co-operating tasks. Each task accesses PVM resources through a collection of library routines. These routines are used to initiate tasks, terminate tasks, send and receive messages and synchronize between tasks. The library of PVM interface routines is one part of the PVM system. The second part of the system is the PVM daemon, which runs on every machine participating in the network for an application and which handles communication between tasks.

PVM can use streams or datagram protocols (TCP or UDP) to transmit messages. Normally, TCP is used for communication within a single machine, and UDP is used for communication between daemons on separate machines. UDP is a connectionless protocol which does not perform error handling. Therefore, when UDP is used for communication, PVM uses a stop-and-wait approach for every packet. Packets are sent one at a time with an acknowledgement for each packet. This scheme gives poor bandwidth on a system with a high latency. An alternative approach is to use TCP directly between tasks bypassing the daemon (Figure 3).

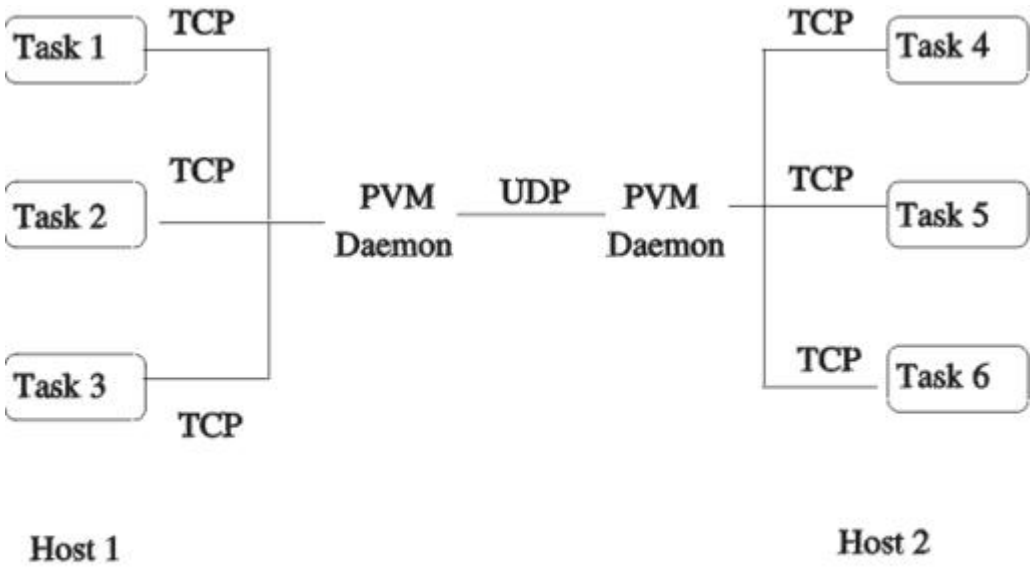


Fig.3 Inter-host communication using UDP

Figure 3. Inter-host communication using UDP

TCP is a reliable transport protocol and does not require error checking after transmitting a packet. The overhead for using TCP is higher than UDP. A separate socket address is required for every connection and additional system calls are used. So, while TCP gives better performance, it is not scalable.

Figure 4. Message Flow within a Host

A number of steps must be completed to send a message from host 1 to host 2 (Figure 4). In the first step, a message buffer is initialized on host 1. Second, the message data is *packed* using the PVM pack routines. Finally, the message is labelled with a message tag and sent to host 2. To receive a message, host 2 issues a PVM call with a message tag and a host ID. Optionally, a wildcard message type or host ID can be used. Host 2 must then *unpack* the message in the order it was packed at host 1.

Experimental PVM implementations using threads and ATM (asynchronous transfer mode) have been developed to obtain a higher bandwidth and lower latency. While the use of PVM is widespread, the Message Passing Interface (MPI) standard is gaining popularity.

What is MPI?

At the Supercomputing '92 conference, a committee, later known as the MPI forum, was formed to develop a message-passing standard. Prior to the development of MPI, each parallel computer vendor developed a custom communication protocol. The variety of communication protocols made porting

code between parallel machines tedious. In 1994, the first version of MPI was released. The advantages of using MPI are:

1. It is a standard and should make porting among machines easy.
2. Vendors are responsible for developing efficient implementations of the MPI library.
3. Multiple implementations of MPI are freely available.

One of the problems with using PVM is that vendors such as Digital, Cray and Silicon Graphics developed optimized versions of PVM which were not freely available. The custom versions of PVM often did not include all the library routines of the PVM system distributed by Oak Ridge, and they sometimes included non-standard routines for better performance. Despite the problems mentioned, PVM is easy to use and developing a new parallel application takes little effort. Converting a PVM application to one using MPI is not a difficult task.

Parallel Programming Concepts

How do you go about parallelizing a program? The first step is to determine which section of the code is consuming the most time. This can be done using a profile program such as **prof**. Focus on parallelizing a section of code or a group of functions instead of an entire program. Sometimes, this may mean also parallelizing the startup and termination code of a program to distribute and collect data. The idea is to limit the scope of parallelism to a manageable task and add parallelism incrementally.

Run with two, four and eight processors to determine scalability. You can expect the improvement in performance to diminish as you use more processors. The measure of parallel performance often used is *speed up*. It is the ratio of the time taken to solve a problem using the best sequential algorithm versus the time to solve a problem using a parallel algorithm. If you use four processors and obtain a speedup of more than four, the reason is often due to better cache performance and not a superior parallel algorithm. When the problem size is small, you can expect a higher cache hit rate and a correspondingly lower execution time. Superlinear speedup is looked upon skeptically since it implies more than 100% efficiency.

While you may obtain good speedup, the results from the parallel program should be similar to the results from the sequential algorithm. You can expect slight differences in precision when heterogenous hosts are used. The degree of difference in results will depend on the number of processors used and the type of processors.

The most efficient algorithm is the one with the least communication overhead. Most of the communication overhead occurs in sending and receiving data between the master and slaves and cannot be avoided. Different algorithms to distribute and collect data can be used. An efficient topology which minimizes the number of communication hops between processors can be adopted. When the number of hops between processors is large, the communication overhead will increase. If a large communication overhead is unavoidable, then computation and communication can be overlapped. This may be possible depending on the problem.

Conclusions

If you are looking for a modest improvement in performance of your application, it is possible to use a cluster of PCs or an SMP (symmetric multiprocessing) machine. But, most applications do not see a dramatic improvement in performance after parallelization. This is true for a cluster of PCs, since the bandwidth and latency are still relatively high compared to the clock speed of processors. For an existing network, there is no additional hardware required and the software to run a parallel application is free. The only effort required is to modify code. Some examples of where parallelism is applicable are sorting a long list, matrix multiplication and pattern recognition.

Manu Konchady (manu@geo.gsfc.nasa.gov) is a graduate student at George Mason University and works at Goddard Space Flight Center. He enjoys flying kites and playing with his 2-year-old daughter.

Archive Index Issue Table of Contents

Advanced search

Parallel Processing using PVM

Richard A. Sevenich

Issue #45, January 1998

PVM is a software application that allows you to turn TCP/IP networked computers into a single virtual machine in order to run parallel programming.

PVM is free software which provides the capability for using a number of networked (TCP/IP) machines as a parallel virtual machine to perform tasks for which parallelism is advantageous. We use PVM in one of our computer laboratories at Eastern Washington University (EWU), mixing Linux and Sun machines together, to comprise various virtual machines. PVM comes with a rich selection of examples and tutorial material allowing a user to reach a reasonable level of proficiency in a relatively short time. There is no new programming language to learn, presuming one already knows C, C++ or FORTRAN. With two or more Linux boxes networked one can run PVM and investigate parallel programming. With only one Linux box, one can still run applications and emulate the parallelism.

PVM was developed by Oak Ridge National Laboratory in conjunction with several universities, principal among them being the University of Tennessee at Knoxville and Emory University. The original intent was to facilitate high performance scientific computing by exploiting parallelism whenever possible. By utilizing existing heterogeneous networks (Unix at first) and existing software languages (FORTRAN, C and C++), there was no cost for new hardware and the costs for design and implementation were minimized.

A typical PVM consists of a (possibly heterogeneous) mix of machines on the network, one being the “master” host and the rest being “worker” or “slave” hosts. These various hosts communicate by message passing. The PVM is started at the command line of the master which in turn can spawn workers to achieve the desired configuration of hosts for the PVM. This configuration can be established initially via a configuration file. Alternatively, the virtual machine can be configured from the PVM command line (master's console) or during run time from within the application program.

A solution to a large task, suitable for parallelization, is divided into modules to be spawned by the master and distributed as appropriate among the workers. PVM consists of two software components, a resident daemon (**pvmd**) and the PVM library (**libpvm**). These must be available on each machine that is a part of the virtual machine. The first component, pvmd, is the message-passing interface between the application program on each local machine and the network connecting it to the rest of the PVM. The second component, libpvm, provides the local application program with the necessary message-passing functionality, so that it can communicate with the other hosts. These library calls trigger corresponding activity by the local pvmd which deals with the details of transmitting the message. The message is intercepted by the local pvmd of the target node and made available to that machine's application module via the related library call from within that program.

Obtaining and Learning to Use PVM

The PVM home page is at http://www.epm.ornl.gov/pvm/pvm_home.html. From there one can download the PVM software, obtain the tutorial, get current PVM news, etc. The software, tarred and zipped, only comprises about 600KB. The README file found therein is sufficient to get up and running, but the value of the tutorial must be stressed. The tutorial can be downloaded from the PVM home page and is also available in book form.

With the tutorial as guide, one can work through the selection of examples packaged with the software. The example source files are well documented internally so that one can become comfortable with the usual PVM library calls. The tutorial has a very nice chapter explaining each of the library calls clearly and in detail. This process was essentially how we started using PVM at EWU.

Our Experience

The network we used at EWU has a variety of machines, architectures and Unix flavors. The first challenge was learning how to configure such a mix of machines to form a PVM. It becomes less straightforward as the heterogeneity increases with the complicating factor being the diverse Unix flavors. One of the methods for configuring a PVM uses the Unix **rsh** (remote shell) command. This relies on the existence of a .rhosts file on the target machine. From the master one starts the PVM daemon (pvmd) on a slave in order to incorporate the slave into the virtual machine—the master uses rsh to do this. However, the target (slave) machine only allows the master access if the master is listed in the target's .rhosts file. The various flavors of Unix do not agree in the finer details for the syntax of the .rhosts file, nor could we always find pertinent documentation. There are several other methods for configuring PVM and, in most cases, we found it a black art. Nevertheless, we persevered and

documented our experience on an embryonic web site for a PVM WebCourse (http://knuth.sirti.org/cscdx/) for those who discover similar problems.

It is instructive to look at an example (see Listings 1 and 2) to get the flavor of the programming. The executable for the source in Listing 1 is started at the command line of the master host which, in turn, spawns the second executable on a slave. For simplicity, no error checking is included. However, PVM provides easy ways to check for failure on such library calls as **pvm_spawn**. The example, adapted from one in the tutorial, merely measures the time it takes for a simple interchange of passed messages, master to slave and slave to master. In particular, the master sends an array to the slave, the slave doubles each of the elements and sends it back. The master prints out the initial array, the final array and the array's round trip time.

Listing 2

PVM has found a regular place in our curriculum. It provides a way to investigate parallel programming in an inexpensive yet realistic way. Thinking through an involved parallel-programming exercise is a new and sometimes difficult evolution for those accustomed only to sequential programming. Once the students have worked through the tutorial material, they move on to problems in which they take a design role and, finally, to substantive projects.

The Future of PVM

Although PVM rather quickly became a de facto standard, critics pointed out that in actuality there was no formal, enforced standard. It was also clear that the message passing was slower than optimized protocols native to an architecture. Very soon after the spread of PVM, a standard was put forward called MPI, the Message-Passing Interface. MPI has advantages (e.g., faster message passing, a standard) and disadvantages (e.g., not interoperable among heterogeneous architectures, not dynamically reconfigurable) with respect to PVM. PVMPI is a newer project which will combine the virtues of both approaches. The success of this project together with any standardization efforts of PVMPI may determine the future viability of PVM.

Resources

Richard Sevenich is a Professor of Computer Science at Eastern Washington University with special enthusiasm for Debian GNU/Linux. His research interests include Application Specific Languages, State Languages for Industrial Control, Fuzzy Logic and Parallel Distributed Processing. He can be reached by e-mail at rsevenich@ewu.edu.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

I'm Not Going to Pay a Lot for This Supercomputer!

Jim Hill

Michael S. Warren

Patrick Goda

Issue #45, January 1998

Los Alamos National Laboratory and Caltech obtain gigaflops performance on parallel Linux machines running free software and built from commodity parts costing less than $55,000 each (in September 1996). Now, you can probably build a similar machine for about $25,000.

As much as we might like to own a supercomputer, high cost is still a deterrent. In a market with almost no economy of scale, buyers find themselves relying on the vendors for specialty hardware, specialty software and expensive support contracts while hoping that the vendors don't join the long list of bankrupt *former* supercomputer vendors. The limited number of sale opportunities force vendors to try satisfying all customers, with the usual result that no one is really happy. There is simply no way to provide highly specialized software (such as a parallelizing compiler) and simultaneously keep costs out of the stratosphere.

On the other end of the market, however, sits the generic buyer. More correctly, tens of millions of generic buyers, all spending vast sums for fundamentally simple machines with fundamentally simple parts. What the vendors lose in profit margin, they make up for in volume. The result? Commodity computer components are increasingly faster, cheaper and smaller. It is now possible to take these off-the-shelf parts and assemble machines which run neck-and-neck with the “big boys” of supercomputing, and in some instances, surpass them.

Why Mass-Market Components?

Intel's x86 series of processors, especially the Pentium and Pentium Pro, offer excellent floating-point performance at ever-increasing clock speeds. The recently released Pentium II has a peak clock speed of 300 MHz, while Digital's best Alpha processors compute merrily along at 500 MHz and higher.

The PCI bus allows the processors to communicate with peripherals at rates in excess of 100MB/sec. Because it is a processor-independent bus, undertaking processor upgrades (e.g., from the Pentium Pros to 500MHz DEC Alphas) requires replacing only the processors and motherboards. Further, parts replaced by an upgrade can be expected to have a significant resale value.

The development of Fast Ethernet technology makes possible point-to-point communication in excess of 10MB/sec. Switches which allow multiple machines to use this bandwidth in full are readily available, which gives the Beowulf-class (see below) machine a bandwidth and latency which rivals the larger IBM SP-2 and the Thinking Machines CM-5. While the Beowulf machines don't yet scale easily to hundreds of processors, their performance in smaller networks of 16 or 32 processors is outstanding.

Free (and well-written) Software

The Linux operating system is robust, largely POSIX-compliant and available to varying degrees of completeness for Intel x86, DEC Alpha and PowerPC microprocessors. Thanks to the untiring efforts of its legions of hackers, auxiliary hardware (network and disk drivers) is supported almost as soon it becomes available and the occasional bug is corrected when found, often the same day. GNU's compilers and debuggers coupled with free message-passing implementations make it possible to use Linux boxes for parallel programming and execution without spending money on software.

Beowulf

The Beowulf Project studies the advantages of using interconnected PCs built from mass-market components and running free software. Rather than raw computational power, the quantities of interest derive from the use of these mass-market components: performance/price, performance/processor and so on. They provide an informal “nonstandard” by loosely defining a “Beowulf-class” machine. Minimal requirements are:

- 16 motherboards with Intel x86 processors or equivalent
- 256MB of DRAM, 16MB per processor board
- 16 hard disk drives and controllers, one per processor board
- 2 Ethernets (10baseT or 10base2) and controllers, 2 per processor

- 2 high resolution monitors with video controllers and 1 keyboard

The Beowulf-class idea is not so much to define a specific system than to provide a rough guideline by which component improvement and cross-platform Linux ports can be compared. Several Beowulf-class machines are in use throughout the United States, including Loki in the Los Alamos National Laboratory's Theoretical Astrophysics group and Hyglac at Caltech's Center for Advanced Computing Research.

Figure 1. View of Loki from the Front

Figure 2. View of Loki from the Back

Loki at LANL

Loki is a 16-node parallel machine with 2GB RAM and 50GB disk space. Most of the components were obtained from Atipa International (www.atipa.com). Each node is essentially a Pentium Pro computer optimized for number crunching and communication:

- (1)Intel Pentium Pro 200MHz CPU with 256K integrated L2 cache
- (1) Intel VS440FX (Venus) motherboard, 82440FX (Natoma) chip set
- (4) 8x36 60ns parity SIMMS (128MB per node)
- (1) Quantum Fireball 3240MB IDE Hard Drive
- (1) Cogent EM400 TX PCI Quartet Fast Ethernet Adapter
- (1) SMC EtherPower 10/100 Fast Ethernet PCI Network Card
- (1) S3 Trio-64 1MB PCI Video Card

The list purchase price of Loki's parts in September 1996 was just over $51,000.

The nodes are connected to one another through the four-port Quartet adapters into a fourth-degree hypercube. Each node is also connected via the SMC adapter to one of two eight-port 3Com SuperStack II Switch 3000 TX 8-port Fast Ethernet switches, which serve the dual purpose of bypassing multi-hop routes and providing a direct connection to the system's front end, a dream machine with the following components:

- (2) Intel Pentium Pro 200MHz CPU with 256K integrated L2 cache
- (1) ASUS P/I-P65UP5 dual CPU motherboard, Natoma chip set
- (8) 16x36 60ns parity SIMMS (512MB)
- (6) Quantum Atlas 4.3GB UltraSCSI Hard Drive
- (1) Adaptec 2940UW PCI Fast Wide SCSI Controller
- (1) Cogent EM400 TX PCI Quartet Fast Ethernet Adapter

- (1) SMC EtherPower 10/100 Fast Ethernet PCI Network Card
- (1) Matrox Millennium 4MB PCI Video Card
- (1) 21 inch Nokia 445X Monitor
- (1) Keyboard, Mouse, Floppy Drive
- (1) Toshiba 8x IDE CD-ROM
- (1) HP C1533A DAT DDS-2 4GB Tape Drive
- (1) Quantum DLT 2000XT 15GB Tape Drive

It is also possible for the nodes to communicate exclusively through their SMC-SuperStack connections as a fast, switched array topology. At Supercomputing '96, Loki was connected to Caltech's Hyglac and the two were run as a single fast switched machine.

Hyglac at Caltech

Like Loki, Hyglac is a 16-node Pentium Pro computer with 2GB RAM. At the time of its construction, it had 40GB disk space, though that has since been doubled by adding a second hard drive of the type listed below to each node.

- (1) Intel Pentium Pro 200 MHz CPU with 256K integrated L2 cache
- (1) Intel VS440FX (Venus) motherboard, 82440FX (Natoma) chip set
- (4) 8x32 60ns EDO SIMMS (128MB per node)
- (1) Western Digital 2.52GB IDE Hard Drive
- (1) D-Link DFE-500 TX 100MB Fast Ethernet PCI Card
- (1) VGS-16 512K ISA Video Card

Each node is connected to a 16-way Bay Networks 28115 Fast Ethernet Switch in a fast switched topology. Video output is directed to a single monitor through switches; the node which is directly connected to the monitor also supports a second Ethernet card and a floppy drive. The list purchase price of Hyglac in September 1996 was just over $48,500. Most of the components have since decreased in price by about 50%, and the highest single-cost item (a 16-port Fast Ethernet Switch) can now be obtained for less than $2500!

Software on Loki and Hyglac

Both Loki and Hyglac run Red Hat Linux on all nodes, with GNU's gcc 2.7.2 as the compiler.

The 200MHz Pentium Pros that drive both systems supply a real-time clock with a 5 nanosecond tick, providing precise timing for message passing. More advanced timing and counting routines are available as well, so that profiling data like cache hits and misses are directly supported. A relatively simple

interface to the hardware performance monitoring counters on the CPU has been developed at LANL called **perfmon**, which is available at the Loki URL listed in the Resources.

Figure 3. Parallel Linux Cluster Logo

Internode communication is accomplished via the Message Passing Interface (MPI). While multiple implementations of MPI are freely available, none was specifically written to take advantage of a Fast Ethernet-based system and, as usual, maximum portability leads to a decidedly less than maximum efficiency. Accordingly, a minimal implementation was written from scratch which incorporated the 20 or so most common and basic MPI functions. This specialized MPI library runs the treecode discussed in the next section as well as the NAS parallel benchmarks for Version 2 MPI, while nearly doubling the message bandwidth obtained from the LAM (Ohio State's version of MPI) and MPICH (from Argonne National Laboratory and Mississippi State University) implementations.

Treecode

Because of its use in astrophysics, Loki has been used to compute results for an N-body, gravitational-interaction problem using a parallelized hashed oct-tree library. (oct-tree is a three-dimensional tree data structure, where each cubical cell is recursively divided into eight daughter cells. treecode is a numerical algorithm which uses tree data structures to increase the efficiency of N-body simulations. For details on the treecode, see the URL listed in Resources.) The code is not machine-specific, so comparing the performance of the commodity machines to traditional supercomputers is free of porting issues (with the exception that the Intel i860 and the Thinking Machines CM-5 have an inner loop coded in assembly).

At Supercomputing '96, Loki and Hyglac were connected via $3,000 of additional Ethernet cards and cables) to perform as a single 32-node machine with a purchase cost of just over $100,000. Running the N-body benchmark calculation with 10 million particles, Loki+Hyglac achieved 2.19 billion floating-point operations per second (GFLOPS), more than doubling the per-processor performance of a Cray T3D and almost matching that of an IBM SP-2 (see Table 1).

Figure 4. Intermediate stage of a gravitation N-body simulation of galaxy formation using 9.75 million particles. It took about three days to compute on Loki at a sustained rate of about 1GFLOP.

As a stand-alone machine at LANL, Loki has performed an N-body calculation with just over 9.75 million particles. This calculation was “real work” and not

“proof-of-principle”, so it was tuned to optimize scientific results rather than machine performance. Even with that condition, the performance and results are striking. The total simulation required 10 days (less a few hours) to step through 750 time steps, performed 6.6x1014 floating-point operations to compute 1.97x1013 particle interactions and produced just over 10GB of output data.

For the entire simulation, Loki achieved an average of 879MFLOPS, yielding a price/performance figure of $58/MFLOP. Contemporary machines such as SGI's Origin are capable of price/performance in this range, but scaling an Origin to the memory and disk necessary to perform a calculation of this magnitude quickly becomes prohibitive; at list price, 2GB of Origin's memory alone costs more than the entire Loki assembly.

The nature of the treecode is such that later time steps have greater overhead in spanning the tree than in performing floating-point arithmetic, so the average flop rate steadily decreases the longer the code is run. When the first 30 time steps of the simulation are taken into consideration, 1.15x1012 particle interactions in 10.25 hours provide a throughput of 1.19 GFLOPS. This figure actually is a better estimate of the amount of useful work than that given for the total simulation, since the treecode's purpose is to avoid floating-point calculations whenever possible.

Vortex-Particle

Loki has also been used to simulate the fusion of two vortex rings. The simulation began with 57,000 vortex particles in two discrete smoke rings, though re-meshing caused the simulation to be tracking 360,000 particles by the final time step. Each processor sustained just over 65 MFLOPS during the simulation for a total system performance of 950 MFLOPS.

Photo-realistic Rendering

Hyglac has been used to perform photo-realistic rendering using a Monte Carlo implementation of the rendering equation. Images of some of the rendered images are available at http://www.cacr.caltech.edu/research/beowulf/rendering.html. In a direct comparison with an IBM SP-2, Hyglac completed the renderings anywhere from 12% to 20% faster than an IBM SP-2, a machine with a price tag twenty times that of Hyglac.

System Reliability

Even the most blazingly fast system is useless if it can't perform without crashing. System reliability is therefore crucial, especially in the case of a machine like Loki which may need several days without interruption to

complete a large-scale calculation. During the burn-in period, a bad SIMM and a handful of bad hard drives were replaced under their warranty terms. The warranties on commodity parts make these commodity supercomputers particularly appealing. Warranties on specialty machines like the Origin tend to be 90 days or less, whereas readily available parts such as Loki's innards generally have warranties ranging from a year to life. In September 1997, most of the Loki nodes had uptimes of over 4 months without a reboot. The only hardware problems encountered have been three ATX power supply fans which failed, resulting in node shutdowns due to overheating. Those nodes were easily swapped with a spare, and the fans replaced in a few minutes.

Price-to-Performance

In Table 2, we summarize the price/performance of several machines capable of running the NAS (Numerical Aerospace Simulation Facility at NASA Ames Research Center) Class B benchmarks: Loki, the SGI Origin 2000, the IBM SP-2 P2SC and the DEC AlphaServer 8400/440.

Conclusions

A gravitational N-body simulation won LANL's Michael Warren and Caltech's John Salmon a Gordon Bell Performance Prize in 1992. A scant five years later, that same calculation can be run on a $50,000 machine. Technology continues to advance (Warren and Salmon recently achieved 170 sustained GFLOPS while running the N-body code with over 320 million particles on half of the nearly 10,000 processors of the Teraflops “ASCI Red” machine at Sandia National Laboratory), but the cost of the ever-improving “high-end” supercomputers keeps them beyond the reach of all but a lucky few. Even those lucky few must compete with one another for processor time in the never-ending game of large-scale computation. Commodity parts provide an opportunity for a handful of users to have a significant share of processor cycles on a machine which is capable of solving enormous computational problems in a reasonable time. Linux and the free software movement provide the software to take full advantage of the hardware's capabilities.

Resources

Jim Hill (jlhill@lanl.gov) is a graduate research assistant at Los Alamos National Laboratory who's thinking about renaming one of his two Linux boxes a zeroth-degree hypercube.

Michael S. Warren (mswarren@lanl.gov) received a B.S. in Physics and in Engineering and Applied Science from the California Institute of Technology in 1988 and a Ph.D. in Physics from the University of California, Santa Barbara in 1994. He is currently developing treecode algorithms for problems in cosmology and hydrodynamics as a technical staff member in the Theoretical Astrophysics group at Los Alamos National Laboratory. He has been involved with parallel computing since 1986 and with Linux since 1993.

Patrick Goda (pgoda@lanl.gov) is currently employed at Los Alamos National Lab in the Theoretical Astrophysics group. When he's not building clusters or hacking Linux software, he actually uses Linux to do real research in Meteoritics (like smashing simulated asteroids into a simulated Earth and seeing how grim the situation would be).

Archive Index Issue Table of Contents

Advanced search

HPF: Programming Linux Clusters the Easy Way

Mike Delves

Issue #45, January 1998

All about high performance FORTRAN and how it is used to write code to run efficiently on parallel computers.

Many programmers use Linux as their operating system of choice when developing applications software. Increasingly, they run the application on the same PC when it is developed. But what do they do if the application becomes too large or runs too long?

The obvious answer is to run it on a cluster of PCs, possibly linked by Ethernet. To do this, you need to write code for each processor. In principle, different code for each, but usually there is common code for all except a selected one or two. The code must include data passing from processor to processor as required. This type of coding uses a “message-passing paradigm” and is in common use on multiprocessor mainframes and workstation clusters. Standard message-passing libraries, PVM (parallel virtual machine) and MPI (message-passing interface), exist to ensure portability.

Writing message-passing code is much harder than writing serial code, and it is easy to show why. Most scientific and engineering programming makes heavy use of one and two dimensional arrays, so we add two vectors this way:

```
DO I = 1,10000
a(i) = b(i) + c(i)
END DO
```

This is clearly a parallel operation: all the elements of **b** and **c** can be added in parallel. For this to occur on a PC or workstation cluster, or indeed any distributed-memory parallel system, the elements of **b**, **c** and, most likely, **a** must be distributed over the available processors. The programmer must arrange this in his code by dealing with bits of each vector on each processor and keeping track of the bits.

A better solution would be for the compiler to do the arranging for you. Then your code could still refer to the complete vector or matrix object and thereby be easier to write, understand and maintain.

The advantages of compiler-aided parallel programming are widely accepted. In the scientific/engineering community, these advantages have led to the development of a standard parallel language called HPF, High Performance FORTRAN, and of HPF compilers for a widening range of architectures.

For good reasons, HPF is deliberately based on the latest, greatly upgraded, version of FORTRAN: FORTRAN95. Beginning with FORTRAN90, FORTRAN contains a rich variety of array facilities so that the loop above becomes the one line:

```
a = b + c
```

which is easier for a compiler, as well as a human, to understand.

Compilers do have difficulty deciding how best to distribute arrays across the processors; thus, HPF gives the programmer a way of providing help. Given that help, the compiler can fully automate the production of a data parallel program from a single FORTRAN77/90/95 source.

HPF codes are portable across SIMD, MIMD shared memory and MIMD Distributed Memory architectures. In particular, you can use HPF on a Linux PC cluster.

The Flavour of the Language

Here is a concocted example to demonstrate the facilities provided by HPF:

```
      REAL a(1000), b(1000), c(1000), &
       x(500),y(0:501)
!HPF$ PROCESSORS procs(10)
!HPF$ DISTRIBUTE(BLOCK) ONTO procs :: a, b
!HPF$ DISTRIBUTE(CYCLIC) ONTO procs :: c
!HPF$ ALIGN x(i) WITH y(i+1)
...
      a(:) = b(:)              ! Statement 1
      x(1:500) = y(2:501)      ! Statement 2
      a(:) = c(:)              ! Statement 3
...
```

The lines starting with **!HPF** are HPF directives; the rest are standard FORTRAN90 and carry out three array operations. What are the directives doing?

The **PROCESSORS** directive specifies a linear arrangement of 10 virtual processors, which are mapped to the available physical processors in a manner not specified by the language. You might expect to need at least ten physical

processors, but most HPF compilers will run the code happily on one or more (up to ten) physical processors. Grids of processors in any number of dimensions up to seven can be defined. They should match the problem being solved in some way—perhaps by helping to minimize communication costs. The processor directive is optional; the number of processors to use can be specified at runtime.

The **DISTRIBUTE** directives tell (actually, recommend to) the compiler how to distribute the elements of the arrays. Arrays **a**, **b** are distributed with blocks of 100 contiguous elements per processor, while **c** is distributed so that, for example, c(1), c(11), c(21), ... are on processor procs(1) and so on.

Note that the distribution of the arrays **x** and **y** is not specified explicitly, while the way they are aligned to each other is specified. The **ALIGN** statement causes x(i) and y(i+1) to be stored on the same processor for all values of i, regardless of the actual distribution.

How the HPF Directives Work

In Statement 1, the identical distribution of **a** and **b** ensures that for all i, a(i) and b(i) are on the same processor; thus, the compiler does not generate any message passing.

In statement 2, there is again no need for message passing. If the **ALIGN** statement had lined up x(i) with y(i) rather than y(i+1), communication would have been needed for some values of i.

Statement 3 looks very much like Statement 1; but the communication requirements are very different because of the different distribution of **a** and **c**. The array elements a(i) and c(i) are on the same processor for only 10 of the possible values of i, and hence for nearly all of the elements; communication of data between processors is needed. This is an unwise choice of distribution for **c**, if indeed this statement represents the bulk of the work.

A good choice of distribution and alignment can greatly help efficiency, and that is the point of having the directives. It is much easier to write FORTRAN90 code and embellish it with HPF directives than to write the equivalent message-passing code.

A Second Example

In practice, the steps taken in writing an HPF program are:

1. Write FORTRAN90 code. Your existing FORTRAN77 code will do in a pinch, but you will get better efficiency by cleaning it up using the newer FORTRAN high-level constructs; tools exist to help this conversion.
2. Decide how to configure the processors.
3. Declare one or more templates to act as guides for distributing arrays.
4. Decide how to distribute and align the arrays onto the template(s).

This process is illustrated in the code shown in Listing 1, which represents a subroutine to solve a set of linear equations. The subroutine is in standard FORTRAN90 and will run happily through any FORTRAN90 compiler, which will treat the HPF directives as comments. The code makes good use of the FORTRAN90 array facilities and has been parallelized by adding just four HPF directives. The resulting HPF code runs well on a Linux PC cluster, provided the size of the problem being solved is large enough to warrant the use of parallelism.

Does It Really Work?

HPF makes life easy for the programmer, by leaving nearly everything to the compiler. So, can the compilers cope? Can you really get parallel efficiency by using HPF? And, can you get useful speedups on networked PCs with relatively high latency communications?

Of course, no compiler can find parallelism where none exists; you need to give it the parallelism in the beginning. Given this, then the answer is yes, current HPF compilers are surprisingly efficient. On a PC cluster connected by Ethernets, the message-passing latency using PVM or MPI is typically around 0.6ms; this translates to “use fairly coarse-grain parallelism if you can and don't expect to use too many PCs.”

Table 1 shows some timings to illustrate what can be achieved. They were taken on a four-PC Linux P100 cluster with 100Mb Ethernet. “Serial” times are those given using the N. A. Software (NASL) FORTRANPlus F90 compiler, release 1.3.57. These times are absent where the code uses HPF extensions (**FORALL**, **EXTRINSIC(HPFSERIAL)**) not supported in FORTRAN90 (for some, we timed equivalent FORTRAN90 versions). HPF times used the NASL HPFPlus compiler, release 2.0. Optimization was set “on” for both FORTRAN and HPF. Times are in seconds.

The overheads intrinsic to using HPF rather than FORTRAN are shown by comparing the Serial and P = 1 times. These overheads are quite low—often

negligible and, for Gauss, even negative (we see this on other platforms too). The gain in using HPF is shown by comparing the Serial and P = 4 times. Speedups achieved relative to the serial times range from 2.1 to 4.5.

Resources

Mike Delves (delves@nasoftware.co.uk) spent twenty-five years at the University of Liverpool as Professor of Computational Mathematics and Director of the Institute for Advanced Scientific Computation. His research interests included numerical methods and their implementation in high-level languages (successively Algol68, Ada, FORTRAN90 and HPF—parallelism crept increasingly in along the way). He started N.A. Software in 1978 as a hobby and is now full-time chairman; the company currently has 23 employees. Linux represents its biggest single market for FORTRAN and HPF compilers.

Archive Index Issue Table of Contents

Advanced search

X-CD-Roast: CD Writer Software

Thomas Niederreiter

Issue #45, January 1998

Mr. Niederreiter tells us all about his graphical user interface for writing data to a CD-ROM.

X-CD-Roast is a fully X-based CD writer program. It is a front end for the command-line utilities **cdwrite** and **mkisofs**. X-CD-Roast therefore reduces the task of creating your own CDs to a few simple mouse clicks instead of a long study of any command-line parameters.

Feature List

The newest version of X-CD-Roast (0.96a in August 1997) provides the following features:

- point and click X11 interface
- automatic IDE and SCSI hardware setup
- copying of ISO-9660 CDs, non-ISO-9660 CDs (like Mac or Sun CDs), Mixed-Mode CDs and Audio CDs.
- mastering of ISO-9660-data CDs
- creation of audio CDs
- quick CD-to-CD copying (no need for a CD image saved on hard disk)
- soundcard support

Limits

At this time, it is not possible to create or copy multi-session CDs with X-CD-Roast. You also cannot master a CD without creating a CD image on a hard disk; you must provide the disk space for the data and for the mastered CD-image. Therefore, you need approximately 700MB times two of free disk space when creating a 650MB CD. Version 0.96a supports only a very limited set of CD writers. The next major update to 0.97 will include the **cdrecord** program which

will handle most writers on the market. Non-SCSI writers will not be supported in the near future.

System Requirements

Prior to the installation of X-CD-Roast you should check whether you satisfy all system requirements.

Required Hardware

How Does It Work?

First, the basic steps when burning a Data-CD: The copying is very simple: first, an image of the original-CD is created on a free hard disk partition (called “image” or “image partition” from now on). This image contains every byte of the original CD and can therefore be up to 650MB in size. Now, you can do a verification run to compare the contents of the image with the original CD. This enables you to track down any read errors that might have damaged the image. The next step is to copy that image back to a CD-Recordable, a process called “burning”. At this point you can choose whether to simulate the burn process (the writer goes through all the motions and processes, but the write laser is off) or to do the real thing. I recommend that you do your first tests using the simulation mode—it could save you a few CD-Recordables.

Audio and mixed-mode CDs (CDs with one data track and additional audio tracks) can be copied as well, but this takes more effort. Each track of the original CD is read into a single file and saved onto the image partition. These tracks can now be written in any sequence you like, and you can also read tracks from more than one CD and so create your own “best-of” audio CD.

If you want to create a CD with your own files, you must first “master” the data. Collect all the files you wish to have on CD in a single directory tree mounted on your system. X-CD-Roast then converts this directory tree into a ISO-9660 CD image. Remember, this image will occupy about the same amount of space on your hard drive as your data to burn does. So, to master 600MB of data, you must have about 1.2GB of space. Finally, you burn this image on to a CD-Recordable.

What is a CD-Recordable?

In this section I briefly explain how a CD-Recordable and a CD writer work. This information is not necessary to use X-CD-Roast, but it can help you to understand the process.

The most common types of CD-Recordables are those with 63 and 74 minutes audio runtimes or 553MB and 650MB data capacities. The structure of a pressed “silver” CD is quite similar to a CD-Recordable. A pressed CD is built from a polycarbonate layer in which the information is pressed as pits. The sequence of “pit” or “not-pit” represents the data on the CD. After the press process, the layer is laminated with aluminum (giving it the silver look) for better reflection of the reading laser of a CD-ROM drive.

On the writable CD, the polycarbonate substrate, has a spiral groove that helps the laser stay on course during burning and playback. On that substrate a thin gold layer is laminated to provide the best possible reflection to a reading laser. Over the gold layer is a photosensitive dye layer, and over that is a clear protective lacquer. While you use a weak laser to read a CD, you need more power to write. The laser beam heats the photosensitive layer and creates a kind of hole that shows us the gold layer. When you read such a CD-Recordable, the holes in the photosensitive layer can be interpreted as “pits”, making it possible to read such a CD with all common CD-ROM and audio players. However, the “pits” are not as sharp as on a pressed CD and can cause troubles on a few older CD-ROM drives, which report occasional read errors, while others perform flawlessly.

Some recommended precautionary measures when dealing with CD-Recordables:

- Label only on pre-marked label side using a soft, felt tip marker.
- Don't use stickers.
- Store only in cool and dark places.

Figure 1. Schematic Drawing of a CD-Recordable

Installation of X-CD-Roast

At the time of this writing (August 1997) version 0.96a is the newest version of X-CD-Roast. You can download it from http://www.fh-muenchen.de/home/rz/xcdroast/. On this web page you will also find the complete documentation and further information. You can also get X-CD-Roast from ftp://sunsite.unc.edu/pub/Linux/utils/ and its mirrors. More detailed information about installation and usage can be found in the X-CD-Roast README file. In this article I discuss only the basics.

First Steps

At this point, I assume that you have installed X-CD-Roast and connected all your necessary hardware correctly. Now start X-CD-Roast as root, and you will be prompted to specify your hardware settings.

Figure 2. Title screen

The Setup Menu

In this menu you specify all the hardware X-CD-Roast needs, including your CD writer, the hard disk partition that will store any CD images, your sound device and so on.

A screen shot showing an example setting for the CD devices is shown in Figure 3. You can always use your CD writer to read data or audio from a CD. It's not necessary to use an additional CD-ROM drive as shown in Figure 3.

Figure 3: Part of the Setup Menu

Data CDs

Use the Copy menu to copy pure data CDs. A data CD is a CD containing only one track written in a certain file system. Usually the file system ISO-9660 is used, but a few special file systems like those from Sun or Apple can also be used. X-CD-Roast is able to copy such a CD independent from the file system by doing a byte-to-byte copy.

First, X-CD-Roast dumps the entire contents of a original CD to the image partition on the hard disk. It is done in this way:

```
cat /dev/sr0 >/dev/sdb4
```

Note that any existing file system on that image partition is overwritten by the CD image. Then, the image is burned to a CD-Recordable using the cdwrite utility.

Figure 4. The Copy CD Menu

Audio and Mixed-Mode CDs

When you wish to copy an audio CD (a music CD) or a mixed-mode CD (a CD that contains one data track and some music tracks), you must copy each track to a single file. X-CD-Roast is equipped with a built-in player so you can listen to each track via a sound card. You can read audio tracks in any sequence and from as many CDs as you wish, as long you keep in mind that only about 74 minutes (or 63 minutes) fit on one CD-Recordable.

The reading of raw audio data from a CD is quite complicated. X-CD-Roast does not simply play a track and record it simultaneously via a sound card, but it does get the audio data via the SCSI bus. X-CD-Roast includes a program **cdda2cdr** which uses the generic SCSI interface to request the audio data from

a CD-ROM drive (or the CD writer) and saves it raw to a file. After you have finished reading in audio tracks, all audio files are burned with the cdwrite utility to the CD-Recordable.

Additional information about the generic SCSI interface can be found in the SCSI Programming HOWTO at ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/SCSI-Programming-HOWTO.

Figure 5. Reading Audio Tracks from a Mixed-Mode CD

The Quick Copy Menu

If you have a CD-ROM drive as well as a CD writer connected, you can speed up the process of copying a data CD considerably. Put the original CD in the CD-ROM drive and a CD-Recordable in the CD writer, and X-CD-Roast will simultaneously read from the CD-ROM drive and write to the CD writer. Thus, there is no need for an image partition, and the time to copy a CD is halved. On the other hand, this procedure can be quite risky. If your CD-ROM drive provides the writer with the data too slowly, you can experience buffer problems that result in a wasted CD-Recordable. For this reason the CD-ROM should be twice as fast as the CD writer. Use this feature with caution.

The Master Menu

If you want to create a CD using your own data, you have to master the data. First you provide a directory tree that contains all the files you wish to burn on a CD-Recordable. This tree can be about 650MB large and mounted anywhere in your file system. (It can even be on a NFS-exported directory.) Now, tell X-CD-Roast how to master the data (see Figure 6) and specify a volume label and, then, start the master process. All your files to burn are now converted into an ISO-9660 CD image that can be burned on the CD as the last step.

X-CD-Roast calls the utility program called mkisofs to do the master job. How long the master process takes depends on the speed of the hard disk, the speed of the processor and the available memory. A 650MB image can be created within 10 minutes on a fast Pentium, but a slower computer can take up to several hours to do the same task. For more detailed information on mkisofs, please consult the man page for this program.

Figure 6. Master: Select Your Image Type

The Future

There you have the rudimentary basics of X-CD-Roast. Keep in mind that X-CD-Roast is just a front end to a lot of utilities, and at this time, is quite limited

compared with the command-line interface of those utilities. I will be developing new versions to add new features such as multi-session on most writers, bootable CD-support, mastering without image-creation on a hard disk and more. Please also check out cdrecord, which can be found at ftp://ftp.fokus.gmd.de/pub/unix/cdrecord. It is command-line based, but supports a lot of CD writers. A future version of X-CD-Roast will use cdrecord instead of the older cdwrite.

For more information about CD writing under Linux, consult the documentation for X-CD-Roast, cdwrite, mkisofs and cdda2wav. I thank all of the great people who put a lot of effort into coding the basic utilities X-CD-Roast uses. Keep up the good work.

An older version of this article was first published in the German language *Linux Magazine*, Issue 11, November 1996.

Thomas Niederreiter (tn@mailserv.rz.fh-muenchen.de) recently finished his study of computer science at the Fachhochschule Munich and now works as an employee in the computer center of that facility.

Archive Index Issue Table of Contents

Advanced search

Netatalk, Linux and the Macintosh

Richard Parry

Issue #45, January 1998

With Netatalk, you can drag and drop files from Linux to Mac and back, share system resources and more.

Unix workstations, PCs running Windows 3.x/95/NT/Linux, FreeBSD, and other systems must be able to communicate seamlessly and share data whether they are in an engineering, business or home environment. More than one type of computer platform existing on the same network is a fact of life. Fortunately, with programs such as Netatalk it's easy to get Apple Macintosh computers and Linux systems to coexist and share resources. This article describes what Netatalk is, what it does, where to get it, and how to install, configure and test it. We also included a short technical description of how Netatalk works.

If you have a Mac and want to run Linux, you can use MkLinux (see *LJ* #31, page 55). If you have a PC and want to run MacOS, you can use Executor (see *LJ* #19, page 40). However, if you have both machines and want to let each do what it does best, use Netatalk. Netatalk lets you keep the operating systems separate while enabling you to transfer files and share printer resources. I paid my dues to wear a Linux T-shirt—I love Linux. I also love my Mac—for entirely different reasons. Since I use the two systems for different purposes, it is often necessary for the two to communicate.

You can transfer small files with a floppy disk (a.k.a. SneakerNet). When the file size exceeds the 1.44MB floppy limit, however, the solution is FTP—fast and efficient, but cumbersome when moving multiple directories containing subdirectories. The ideal solution is to mount the Linux file system on the Mac desktop, then drag and drop files and directories as you do under the normal MacOS. This is exactly the solution that Netatalk gives you. You can also use Netatalk to send print jobs to the Linux printer from the Mac, thereby sharing other system resources.

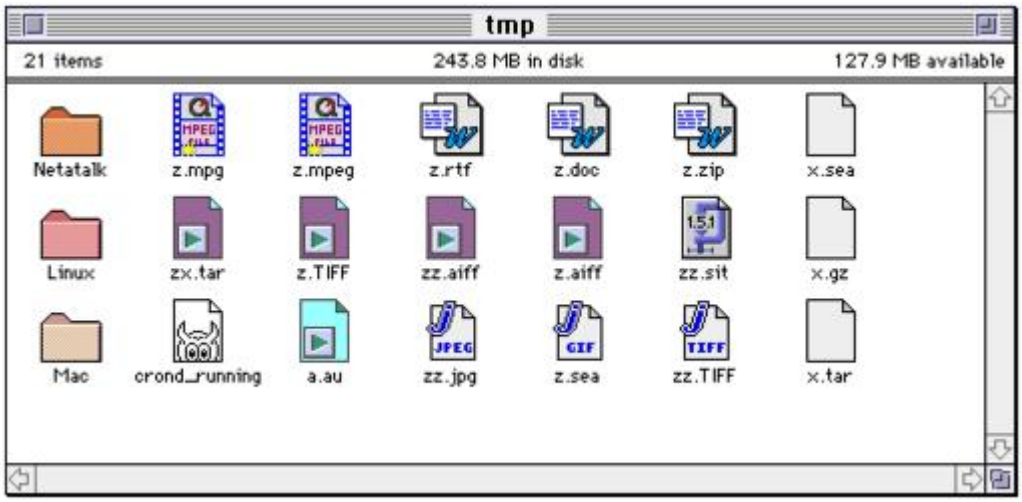


Figure 1. Linux Files and Directories on the Macintosh Desktop

At the time of this writing, the most recent version of Netatalk is 1.4b2. Don't let the beta suffix scare you; Netatalk is stable, especially for the PC platform. Current development emphasizes new platforms (see Table 1) such as FreeBSD and Solaris—hence the beta appellation. The primary Netatalk host site and directory is: ftp://terminator.rs.itd.umich.edu/unix/netatalk/. From that address all you need to do is download the single file netatalk-1.4b2.tar.gz. You can uncompress and untar Netatalk with normal Linux commands such as:

```
gunzip netatalk-1.4b2.tar.gz
tar xvf netatalk-1.4b2.tar
```

or:

```
tar xzvf netatalk-14b2.tar.gz
```

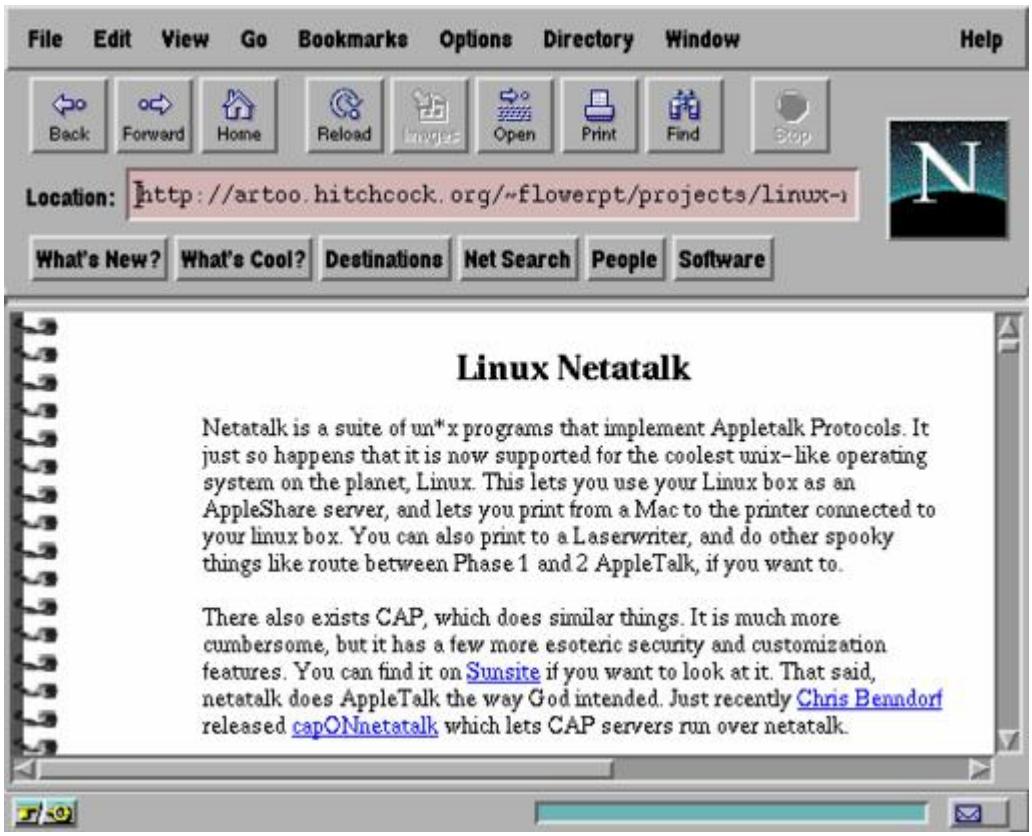


Figure 2. One of Several Netatalk Web Pages

Note that prior to the release of Linux 2.x, installation of Netatalk required that the Datagram Delivery Protocol (DDP) software be integrated into the kernel. Now that DDP is part of the Linux kernel, all you need to do is make sure that you turn on Appletalk during kernel configuration. Do this by specifying the kernel configuration option as:

```
CONFIG_ATALK=y
```

You should always use the README files that accompany the Netatalk distribution as the ultimate authority for installation instructions. Since Netatalk is offered for several platforms, as shown in Table 1, there are actually several different README files. Each operating system has its own defaults for file locations. For this reason, the Makefile that comes with Netatalk provides user-defined variables that you can change to alter the installation and ultimately the file structure. For the Linux installation, I used the default values and the installation worked flawlessly.

The **DESTDIR** variable in the Makefile points to the directory where binaries are placed. The default for this directory is /usr/local/atalk and does not need to be altered. Note that setting **DESTDIR** causes all installation-relative pathnames to be set correctly. The other important variable is **MANDIR**, by which you can specify the location of the man pages. The default for this directory is /usr/man, and again does not need to be changed.

Installation to make all binaries begins with the ubiquitous command:

```
make
```

To install the binaries at the root of the source tree, type:

```
make install
```

Pretty easy! Now let's go on to configuration.

Configure Appletalk

First, add the contents of the services.atalk file that accompanies the Netatalk distribution to your /etc/services database. The lines to add will look like this:

```
rtmp    1/ddp   # Routing Table Main Protocol
nbp     2/ddp   # Name Binding Protocol
echo    4/ddp   # AppleTalk Echo Protocol
zip     6/ddp   # Zone Information Protocol
```

Next, assuming that your physical connection between Linux and the Mac is Ethernet, edit the /usr/local/atalk/etc/atalkd.conf file so that it contains the single line:

```
eth0
```

This line is automatically changed when the system is placed into service. In my case, when I actually connected to the network, the **eth()** line (the file configuration) was changed by the Appletalk daemon to:

```
eth0 -phase 2 -net 7540-7544 -addr 7544.197 \
        -zone "Outer-Limits"
```

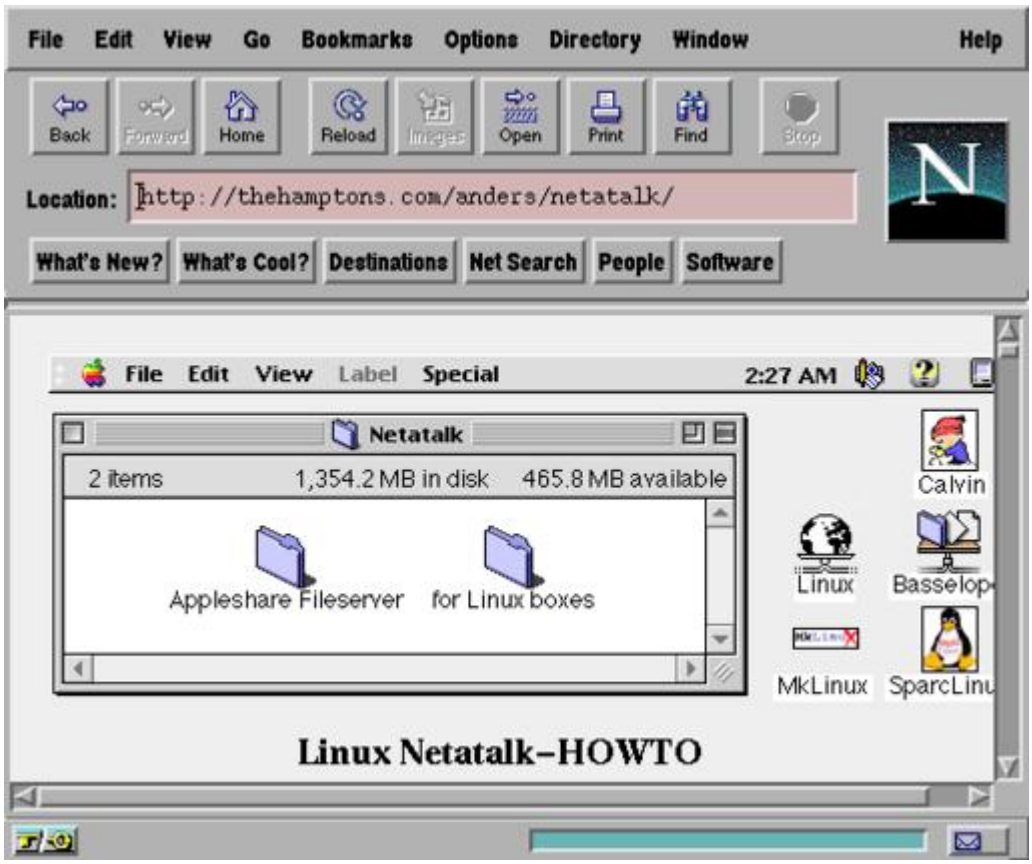


Figure 3. Your eyes aren't deceiving you—It's a web site devoted entirely to Netatalk.

Exporting Linux

The Netatalk distribution provides three sample configuration files that need to be configured:

1. AppleVolumes.default
2. atalkd.conf
3. papd.conf

These files are normally placed in the /usr/local/atalk/etc directory, and contain instructions in the form of comments to aid the user. You can get additional information from the man pages. We have already discussed the configuration of atalkd.conf during Appletalk configuration.

You specify which files and/or directories are made accessible to the Macintosh by editing the file /usr/local/atalk/etc/AppleVolumes.system. Here's a sample configuration is shown in Listing 1. This configuration shows that the directories /tmp and /pub are available to the network. On the Macintosh a Linux directory appears as a folder with a name you assign: in our example tmp and pub. The directory /home/user/X is also accessible from the Macintosh under the name XWindows. You specify that an exported directory will have a different name on the Mac, using a different second parameter than the Linux directory name, such as /home/user/X XWindows.

We advise caution when deciding which directories to export to the network. The Netatalk daemon **afpd** (Apple Filing Protocol Daemon) creates additional files for housekeeping. Specifically, in each of the exported directories, two invisible files (that is, files beginning with a period character) are created, namely, .AppleDesktop and .Appledouble. These files are created in all Linux directories and subdirectories that are exported and accessed. If you export an entire file tree using the / character, you will find your Linux system cluttered with .AppleDesktop and .Appledouble files.

Also, note that the file /usr/local/atalk/etc/AppleVolumes.system contains a list of file extensions. This list is not necessary, but the extensions provide a means for displaying the appropriate icons on the Macintosh desktop. For example, in Figure 1 you can see that the files z.mpg, z.rtf and zz.jpg are displayed with the correct Macintosh application icon.

Sharing the Linux Printer

You can make a printer connected to your Linux system available to the Appletalk network by using **papd** (the Appletalk Printer Access Protocol Daemon), which is started when booting Linux. The papd daemon accepts printing requests from the Macintosh and spools the job to the local line printer. To configure the printer, edit the file /etc/local/atalk/etc/papd.conf to look like this:

```
# Sample papd.conf file to allow
# printing on the Linux printer over
# the Appletalk network.
MyLaserW:\
        :pr=lp:op=cg:
```

The syntax used in this file is the same as used in the Linux printcap file that you can find on most Linux systems in /etc/printcap.

Netatalk Startup

The file rc.atalk located in the directory specified by the variable **ETCDIR** (normally /usr/local/atalk/etc) must be executed at startup. To do this on a Linux system on which the rc files are to be kept in one place, copy /usr/local/atalk/etc/rc.atalk to the /etc/rc.d directory or make a symbolic link. Then, using a text editor, insert the following line in the rc.local file:

```
sh /etc/rc.d/rc.atalk
```

For more information about this script, consult the Netatalk man pages.

Testing

Now for the fun part—go to the Macintosh Chooser and select AppleShare as shown in Figure 4. The Linux system host name will appear on the desktop just like any other Appletalk-compatible machine on the network. There is an 8-character password limit on the Mac, so if your Linux password is more than 8 characters, change it.

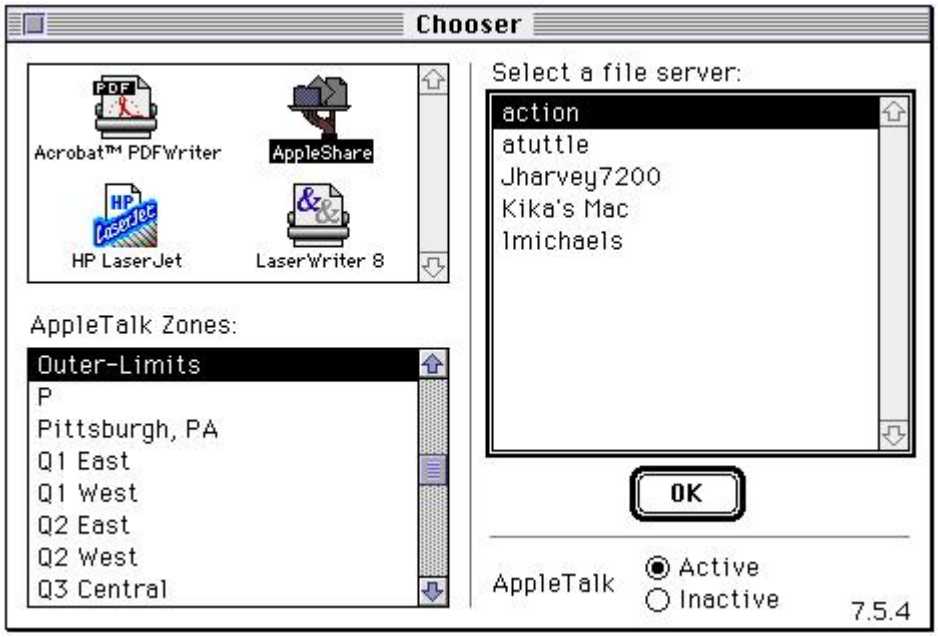
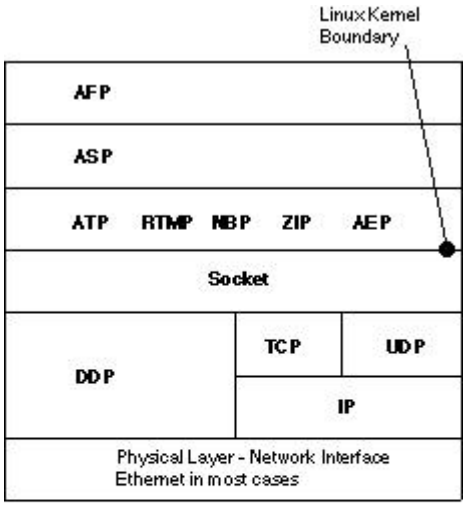


Figure 4. Use the Mac Chooser to Mount the Linux File System

Netatalk Techtalk

Technically speaking, Netatalk is an implementation of the AppleTalk Protocol Suite. It contains support for EtherTalk Phase I and II, DDP, RTMP, NBP, ZIP, AEP, ATP, PAP, ASP and AFP, as shown in Figure 5. DDP is now provided by the new Linux 2.x kernel. The **atalkd** daemon implements RTMP, NBP, ZIP and AEP, which is the AppleTalk equivalent of Unix **routed** (route daemon). ATP and ASP are implemented as libraries. The **papd** daemon allows Macs to spool to **lpd** (line printer daemon), while **pap** allows Unix machines to print to AppleTalk connected printers. Also provided in the suite is **psf**, which is a PostScript printer filter for lpd designed to use pap. A PostScript reverser, **psorder**, is called by psf to reverse pages printed to face-up stacking printers. Last and perhaps most important is the afpd daemon that provides Macs with an interface to the Unix file system.




```
AEP  - AppleTalk Echo Protocol
AFP  - AppleTalk File Protocol
ASP  - AppleTalk Session Protocol
ATP  - AppleTalk Transaction Protocol
DDP  - Datagram Delivery Protocol
IP   - Internet Protocol
NBP  - Naming Binding Protocol
RTMP - Routing Table Maintenance Protocol
TCP  - Transport Control Protocol
UDP  - User Datagram Protocol
ZIP  - Zone Information Protocol
```

Figure 5. The Netatalk Stack

There are extensive and well-written man pages that accompany Netatalk, such as aecho.1, afpd.8, atalk.4, atalkd.8, atalk_aton.3, getzones.1, hqx2bin.1, macbinary.1, megatron.1, nbp.1, nbplkup.1, nbprgstr.1, nbp_name.3, pap.1, papd.8, papstatus.1, psf.8, psorder.1, single2bin.1, unbin.1, unhex.1 and unsingle.1.

Conclusion

Netatalk is a stable program that makes moving files between the Mac and Linux as easy as drag-and-drop. In fact, you can install Netatalk on almost any Unix-like platform and take advantage of the power it provides. Now you really can have the “power to be your best” by using both the Mac and Linux.

Resources

Richard Parry (W9IF) is currently attending the University of California, San Diego and studying computer science. He works as a software engineer at Qualcomm, Inc., known by most as the home of Eudora. His wife tells us that he spends entirely too much time with his Linux system. He can be reached via e-

mail at rparry@qualcomm.com or you can visit his home page at http://www.qualcomm.com/~rparry.

Archive Index Issue Table of Contents

Advanced search

LJ Interviews Mike Apgar, Speakeasy Café

Marjorie Richardson

Issue #45, January 1998

Speakeasy was conceived around March of 1994. The main impetus was that while the Internet was fascinating to me, I found the necessity of access only at home to be both socially debilitating and much too slow.

I interviewed Mike Apgar, owner of the Speakeasy Café in Seattle, by e-mail on July 15. I not only got answers from Mike but also from Chris Osburn, Speakeasy Network Administrator. Mr. Apgar has a varied history of work in computers as a programmer and analyst. He started Speakeasy with a team of people including an architect, a designer, an attorney, a systems administrator, a playwright and a business executive (his dad).

Margie: Let's start with a few questions about Speakeasy. What factors convinced you to open a cybercafé?

Mike: Speakeasy was conceived around March of 1994. The main impetus was that while the Internet was fascinating to me, I found the necessity of access only at home to be both socially debilitating and much too slow.

One evening a friend was visiting me, and he needed to find some specific information about a health topic for a play he was writing. We had a great time searching the Web and discovered that the Web can be a lot more fun when you're browsing it with other people.

One day, not more than a week later, I heard an interview with William Gibson in which he mentioned SFNet (the network of public access, “local-net” at the time) terminals.

I thought of creating a café at that very moment—one that would be a café as you usually think of one: a casual place to have a cup of coffee, not necessarily “computer-centric”.

My wife Gretchen and I talked about it that evening and decided on the name “speakeasy”—we loved the double entendre and had always held a certain fascination for the old speakeasies of the prohibition era.

Figure 1

We knew that it would be about people connecting but had no idea what a success it would be, only that we would run a “nice little café” and “do our own thing”.

Later, months before opening, we got more ambitious—please see our mission statement at http://www.speakeasy.org/speakeasy/cafe/mission.html.

Margie: When did you open? your location?

Mike: We opened in June of 1995 at 2304 2nd Ave in Belltown, downtown Seattle.

Figure 2

Margie: What is your repeat customer percentage? How big is your regular clientele?

Mike: Our repeat customer rate is very high, though I don't have a specific percentage. We probably have around 800-1000 very regular customers—meaning at least once a week.

We believe there is a natural tendency for people to form a community in order to trade ideas, tips, secrets, hints and simply communicate about collective situations. Internet computer technology is a powerful tool that helps create such a community. This very thing has happened twice before in this century: first with radio, next with television. As we live in an ever more pressing age for people to communicate and resolve, reason and understand one another, our efforts are designed to facilitate that information exchange.

Margie: Do you get a pretty varied clientele? Any particular age group or profession that come more than others?

Mike: It seems to be weighted in a couple of directions—twenties mostly, with a large number of people “in multi-media fields” as well as people in computers and the arts in general. Another large part of our customer base is more of the thirties and forties crowd that are professional and often net-savvy.

In general, our service appeals to people of all ages. For example, the virtual tours we hold on a regular basis have wide appeal to people who do not

associate themselves with the computer age. These classes and special events showcase and educate users on specific opportunities on-line and examine the current and potential roles for technology in our communities.

Figure 3.

The café is a place for people to gather at any hour. People stop in before work to get their espresso. While waiting, they read news from around the world, send e-mail to a friend across the country or post an article to the local Speakeasy newspapers, electronic magazines or classified advertisements. After work they drop by to read through personal ads, send a bit of electronic mail to the mayor or vote on future Speakeasy events.

Also, in the evenings we draw a large variety of customers, young and old, for our entertainment: the art gallery, live music, poetry readings, film or readings —most happen once a week. The Art Gallery, curated by Tina LaPadula, rotates a new show every 6 weeks.

Margie: How many computers do you have?

Mike: We have 9 graphical workstations and 12 text-only "VT100" style terminals —that's 2 PCs, 1 Mac and 6 X terminals. All graphical terminals have 17-inch color monitors.

We are bringing Windows applications to the X-terminals through a new server application called Win-Center by NCD. This is currently in beta—we're hoping to provide access to the major Windows applications that don't have a Unix counterpart, such as MS-Office and Adobe Photoshop. We are, however, planning to offer WordPerfect on the X-terminals as soon as possible, perhaps the entire Corel Office Suite.

Margie: How do you charge for computer time? If you have a monthly membership rate, how many people take you up on it?

Mike: Our graphical terminals are anywhere from $6-10 per hour, depending on whether you are a Speakeasy member. The PCs and Macs have a higher hourly rate.

Members can buy "bulk-time" and get rates as low as $3.50/hour. Monthly membership is $10/month, $50/6 months or $90 for a year and includes an e-mail account (mike@speakeasy.org), 10MB of disk space for non-commercial web pages or simply file storage, *free* use of our text-only terminals and access at all RAIN sites (Remote Access Internet Nodes). For more details on RAIN, see our web site at http://www.speakeasy.org/rain/.

Margie: What kind of services do you offer? I hear the Pacific Northwest Ballet has a web page set up through you.

Mike: In short, we provide: Web Design, Web Hosting, Dedicated Connectivity (Modem, ISDN), Firewalls, Mail and File Servers (often Linux), LAN Integration and general LAN assistance and quite a bit of general small business consultation related to the Internet.

For even more details, I'd kindly refer you to http://www.speakeasy.org/speakeasy/network/, and our own web site at http://www.speakeasy.org/.

We provide some or all of the above services to over 300 commercial clients/organizations including: The Seattle Symphony, Virginia Mason, Pyramid Brewery, The Seattle Opera, Washington CEO/Fivash Publishing, Media Index/Media-Inc, KeyStaff, LifeSpan BioSciences, Sonus Pharmaceuticals, Bumbershoot, AT&T Summer Nights, AT&T Family Fourth, Puget Sound Community School and many many more—our full client list is available at http://www.speakeasy.org/speakeasy/clients/.

Chris: And yes, the Ballet, at http://www.pnb.org/.

Margie: I understand you are using the Linux operating system. How are you using it and why did you choose it?

Chris: We currently have four Linux servers in the café. We use them for internal routing, Usenet and WWW service. We are using a couple as X workstations—one runs SAMBA to serve files to the inevitable Windows PCs. We chose Linux because of its price and its reputation for stability.

One exciting application is the RAIN sites, mentioned earlier. RAIN sites are terminals we've placed in other coffee shops and public locations around Seattle. At RAIN sites with more than one terminal, we've placed a small 386 with a multiport serial card to act as a terminal server. Over time, we've actually gotten everything we need to fit on one floppy, eliminating the need for a hard drive. The whole OS and necessary services run on a RAM disk.

We typically specify Linux when setting up mail servers and firewalls for clients, mainly to ease remote maintenance of these resources.

And in the café, our developers are so CPU-hungry that NT may as well be “Not There”.

Margie: Give us the details of how you have the system configured. Are you using the Apache web server?

Chris: We generally start with Slackware, though we've been using Red Hat as well. We do use Apache for our web servers. Again, the price is right, and we can modify the server for our own needs.

Margie: Does Linux work well for you? How do you compare it to other operating systems that you have used?

Chris: I find Linux easy to install, fast in execution, and very well supported. (I may be biased—I learned Unix well before Microsoft got the DOS contract. As far as I'm concerned, they've been playing catch-up ever since.) The operating system is extremely stable. One system, primarily used as a development machine, but also serving as an internal router, web server and development box, has been up 160 days as I write this. We've actually been putting off hardware upgrades, waiting for it to crash.

Margie: Do you feel cybercafés are here to stay or will they go the way of other fads?

Mike: Public Internet Access is definitely here to stay. Many “cybercafés” (we don't describe ourselves that way, by the way) are trying to build a business on the novelty of high-speed Internet access or just plain Internet access in a café, and there is no more of a certain future in that approach than in any other restaurant/café business, I'm afraid—probably even less.

From the beginning, we planned for the long-term. I think the variety of services we offer, and the atmosphere of the café describes that well. It seems to have worked for us; we're beginning our third year in business and are growing very fast.

Margie: What role do you think Linux will play in the future of cybercafés?

Chris: I don't know about a role for Linux in cybercafés in general—however, I can speak for our business. Linux continues to play an ever larger role in everything we do. Look for us to bring Linux forward as a part of our advertising and marketing efforts. It's very fast, very stable and growing in sophistication many times faster than other versions of Unix and certainly has more to offer at this point than other popular operating systems.

Mission Statement

Archive Index Issue Table of Contents

Advanced search

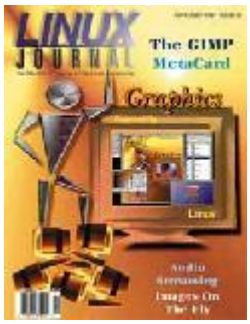
LINUX JOURNAL

The Quick Start Guide to the GIMP, Part 3

Michael J. Hammel

Issue #45, January 1998

This month we learn how to use the Image Window and layers in building our images with the GIMP, a Linux power tool for the graphics artist.



Patience and persistence has paid off. After two articles you sat and wondered “When do we get to the meat?” In the first article of this series I introduced the GIMP and explained some basics about retrieving the program and getting it running. Last month I took a look at some of the GIMP's windows and and described the image file formats it supports. We're now ready to start looking at how you can use the GIMP. I'll start by looking at where images are displayed and manipulated: the Image Window.

The Image Window

Figure 1 shows an Image Window along with the Layers and Channels dialog. The visible parts of the Image Window consist of the Image display area, the rulers across the top and left sides and the scrollbars across the bottom and right sides. Pop down menus and the Ruler Guides are not visible by default but are easy to access.

Figure 1. Image Window with Dialog Box

By default, if the Image Window's dimensions fit into the displayed screen area, the rulers are divided by units of 10 pixels with major tick marks at intervals of

100 pixels. There are two other sizes of tick marks, those for units of 20 pixels and those for units of 10 pixels. If the Image Window's dimensions are larger than the displayable screen area, the GIMP will provide a window that uses major tick marks at units of 250 pixels. However, since the image would actually be larger than the screen, the Image Window does not display the image pixels one per display pixel. Don't let this confuse you. You can zoom in on your image to get pixel-level details. The GIMP is just smart enough to know how to handle images that are too big for the display. This capability is useful for design work destined for print media.

The units represented by the tick marks in the rulers can be specified in the gimprc file (which was covered in the first article in this series). By default the units are in pixels, but this can be changed to inches or centimeters at the user's discretion.

Guides are straight dashed lines alternating red and black. These extend from the left to right edges of the Image Window's display area or from the top to the bottom of the display area. To create a new Guide, move the cursor over a ruler (top or left side), hold down the left mouse button and drag into the display area. A new Guide, parallel to the ruler from which you dragged the cursor, is created. When the cursor is over the Guide, it changes to a left-hand pointer (a hand with the index finger extended). Hold down the left mouse button to grab the Guide and move it to a new location. To remove the Guide, grab it and drag it back over the ruler and release the mouse button.

Figure 2. Two Guides

Figure 2 shows two Guides, one horizontal and one vertical. Guides are useful for aligning objects or for snapping selections to specific areas. You can toggle selections to snap to the Guides from the Image Window's pop-down menu ("View->Snap" To Guides). Guides, although visible in the Image Window, are not saved to files of any image type except XCF. If you save a file to GIF or JPEG or TIFF for example, the Guides will not show up in the image file.

The scrollbars perform just as scrollbars in any other windowing application. Grab the slider with the left mouse button held down to slide the image up/down or left/right. Click in the scrollbar outside of the slider (if the slider does not extend the length of the scrollbar) to page through the display area. Click on the arrow buttons on the ends of the scrollbar to increment or decrement your way through the display area.

The most important features of the Image Window are the pop-down menus. From these you will have access to all of the GIMP's image processing capabilities. To post (i.e., display and leave open) the menus, place the cursor

over the display area and click the right mouse button once. The top level menu will open and stay open until you either click outside the menu windows or select a menu option. Some of the options open submenus, which will also stay open if you simply click on those options. Clicking/dragging over the menus also works, although the menus will close if you drag the mouse off the menus and release the mouse button.

The Image Window's pop-down menus are divided into the following categories:

- **File Menu**: new, open, close (window), quit (completely), etc.
- **Edit Menu**: copy, cut, paste, clear, stroke, etc.
- **Select Menu**: all selection methods, including growing, feathering and shrinking selections
- **View Menu**: zoom in/out, toggle rulers/guides, etc.
- **Image Menu**: invert, color balance/contrast/adjust, convert to/from RBG/ Grayscale/Indexed, etc.
- **Layers Menu**: functions applicable to the currently active layer.
- **Tools Menu**: access to all the tools available from the Toolbox.
- **Filters Menu**: the large set of image filters such as blurring, embossing, bump mapping and many more.
- **Dialogs Menu**: provides a convenient way to open the various dialogs such as the Gradient Editor, the Layers and Channels dialog, the color palette and the brushes dialog.

Descriptions of all of these would be far more than I can fit into one series of articles. I will cover some of the more interesting filters in next month's article and will provide a detailed description of the Toolbox. Most icons accessible from the Toolbox are also accessible from Image Window menus, and those that aren't are modifiable through Image Window menu options.

The rest of the menu options each have fairly obvious meanings. The File Menu options are nothing special, although you may want to take a look at the Preferences... option. Selecting this option opens a window which provides the user with access to configuring the way transparency is represented in layers and in the gradient editor. These changes are for the current session only and are not saved across invocations of the GIMP. You must manually edit the gimprc file to make permanent changes to these preferences.

An interesting option under the Edit Menu is the Stroke option, available after a selection is made. I'll discuss selections more in a moment, but for now, note that there are many ways to select portions of your images in the GIMP, and each method is a form of a selection. A selection is denoted by a dashed line

around the edge of the selection that appears to move around that edge and is known as “marching ants”. The selection outline can be used to draw a line along that edge by using the Stroke option. The color of the line must be chosen after the selection is made, and you should also select the appropriate brush. Once that is done, just select “Edit**->**Stroke” and a line in the selected color will be drawn using the selected brush. I know, I know—“How do you choose the color?”, “How do you choose the brush?” That's next month—there's just so much to cover.

You will find that one of the most important operations you will use with the GIMP is that of selecting portions of the image for modification. There are rectangular, ellipse, free-hand, fuzzy and bezier selection tools available from the Toolbox. The Image Window's Select Menu offers other methods such as all, none and select by color. This last method is one I use quite often to accurately select regions of irregularly shaped (but uniformly colored) areas of an image. I can then keep that selection, add a new transparent layer and fill the selection with a gradient in the new layer. Selections can be made in any layer. The selection doesn't actually affect particular pixels. They just mark a region of the image. Because of this, you can change which layer you are working on after you've made a selection and then work with the same selected region in another layer. This will allow changes to the new layer within the selected region and leave the old layer, where the selection was originally made, unchanged.

The GIMP is an image processing tool. It works by manipulating images based on the color, distribution, frequency and other aspects of the pixels which make up those images. I haven't talked before this about the sorts of manipulations that can be done. A great majority of them are done through the use of *Filters*, also known as *Plug-Ins*. This is a large area of discussion and better suited to the last article in this series, however not all image manipulation is done through Filters. The Image Menu in the Image Window's menu allows users to do a number of image processing tasks on an image and/or its layers. Most of the image processing tasks here relate to the general color, brightness or tint to an image as a whole and worry less about how individual pixels relate to their neighbors.

The Image Menu also allows transforming the image type from RGB (generally known as high color, 24-bit images) to Indexed (256-color images) to Grayscale (black and white) images. If you find you want to change a color image to black and white, but leave it as an RGB image (so you can add color to it later), you can use the “Image**->**Adjust**->**Desaturate” option. You can also adjust the amount of Red, Green or Blue in an image using the “Image**->**Adjust**->**Curves” option. I've often used the Curves and “Image**->**Adjust**->**Color Balance” option in conjunction with the Chrome Script-Fu script to add or enhance color to the

edges of text logos. The latest version of the *Linux Gazette* logo was created in part using this technique.

Layers

I just got through mentioning the use of layers in conjunction with selections. Layers are like transparencies used for overhead projectors, with the ones on top mixing with layers below to produce the final image. Each layer can be added, subtracted, multiplied, screened, overlayed and have any of a number of other processes applied to it with respect to the layers below it. By default, a new image has one layer—the Background layer. The default for this layer is to be solid white, but it is possible to change this based on the background color and transparency settings.

Layers are accessed via the Layers and Channels dialog box. This dialog is opened from the "Dialog->Layers and Channels" menu option from the pop down menus in the Image Window. Figure 3 shows a sample of the dialog in use. Inside the dialog there are a number of features: an image selection menu, the Close button and a notebook with tabs for choosing either the Layers or the Channels feature. In the Layers page of the notebook there is a menu for selecting the Mode to apply to this layer (such as add, subtract, darken, etc.). It's also possible to apply a uniform transparency level to the current layer using the Opacity slider.

Figure 3. Layers and Channels Dialog Box

Below the slider there is a window which shows each of the layers for the current image specified in the Image Menu (which is located at the top of the Layers and Channels Dialog). Each layer displayed contains four default fields: visibility, move anchor, a thumbnail preview of the layer and the layer's name. A grayscale mask field is not displayed by default, but can be created by selecting the Add Layer Mask option from the Layer Menu. This menu is opened by clicking the left mouse button over the name of the current layer and is used for all layer specific functions except transparency and mode selection. Double clicking on the layer's name will bring up the Edit Layer Attributes dialog in which you can specify a new name for the layer. Get into the habit of setting the names of layers often, since many of your images may have many small layers; finding the one you want simply by looking at the thumbnail may not be easy. In most cases, changes to an image affect only the currently active layer. The active layer is the highlighted one in the list of layers. On most systems this is a blue highlight, although it's possible this might be some other color on very low-end systems with limited color displays.

A few things to note. First, changing the name of a layer will add an Alpha channel to the layer. There are lots of special tricks you can do with Alpha

channels that are beyond the scope of this series, but you should be aware that the Background layer, by default and before you have changed its layer name, has no Alpha channel. Also, the size of the thumbnails can be set to just about any size via an option in the gimprc file. As was mentioned in the first article, you should eventually become familiar with the gimprc file, if you find the default settings are not to your liking. You'll be surprised at how much you can configure in the GIMP.

The visibility icon, an eyeball, is used to turn the layer on or off. If the layer is off, its effects on the visible image in the Image Window are removed. Clicking on the field containing the eyeball will toggle the visibility of the layer. When the visibility is off, the eyeball will not be displayed. The visibility icon will be in the far left side of the currently selected layer, directly beneath the “Opacity” text in the Layers and Channels dialog. If you don't see the eyeball there, click the left mouse button in that area and the eyeball will appear. This may or may not affect the visible image in the Image Window, depending on the contents of that layer and the Mode currently selected.

Often it is useful to move more than one layer at a time, i.e., as a group. By default using the Move Tool (in the Toolbox) will affect only the current layer. Clicking on the second field for a layer, the move anchor will set that layer to move along with any other layers which also have their move anchor set. When the move anchor is set, a *fleur* icon is displayed. A fleur consists of two double ended arrows, perpendicular to each other. Although actions on an image specifically affect the currently active layer, any layer with its fleur displayed will also be moved when the Move Tool is used on the current layer.

Moving a layer can take a little practice. Some layers, due to the use of transparency, are harder to grab than others because very little of the non-transparent region is actually visible. Other layers, higher in the stacking order, may obscure most of a lower layer. When moving a layer you will find it easiest to first select the Move Tool from the Toolbox. Next, move the cursor over a region in the Image Window that you suspect is part of the layer you wish to move. The cursor should change to the fleur symbol when you are over that region. If you then click on that spot and discover the layer selection has changed to another layer, then stop. Press **ctrl-Z** to undo any moves you've made, if any. (If you accidently undo something you didn't want to, just use “Edit**->**Redo” in the Image Window menus or **ctrl-R** to get it back.) Now turn off the visibility of the higher level layers until the layer you want to move is the highest visible layer. You should now have a better chance to grabbing just the layer you want. If you still have problems, you can turn off the visibility of all the layers, higher and lower levels. Again, all of this takes a little practice.

The list of layers is shown with the topmost layer at the top of the list and the bottom layer, the background layer, at the bottom. When the layers are combined, also know as *composited*, they are combined from bottom to top. So the bottom layer defines the start of the image's initial appearance before the other layers are combined with it. The Image Window will display the image based on which layers are currently visible. The compositing is done by the GIMP in a way that is invisible to the user except for changes made to the Mode or visibility of a layer. Since the layers are processed from bottom to top, the order of the layers is important. Layers can be raised or lowered using the Layers Menu. The Layers Menu is opened by clicking the left mouse button over the name of a layer. Selecting an option in the Layers Menu only affects the layer over which the menu was opened.

The size of the preview in the Layers and Channels dialog can be set in the gimprc file. Smaller preview sizes do not really speed up processing but do require less memory and allow more layers to be displayed simultaneously. Layers themselves are not constrained to the same size, although when producing the final image, a constraint of some kind will be applied to all layers. The constraint used depends on the way the final image is produced from the layers.

Layers provide tremendous flexibility in producing images in the GIMP, but they can cause a little confusion when first starting out. It is important to note the following about layers when saving or loading images:

- New images have a single, white background layer.
- Opening an existing file in any format except XCF will load the image into the background layer. You cannot open an image into a layer of an existing Image Window. If you want to do this, you need to open the image in a new window, select the entire image (**ctrl-A** in the Image Window), copy it (“Edit**->**Copy”) and then paste it (“Edit**->**Paste”) into the existing Image Window.
- Saving an image in any format except XCF will save only the currently active layer. If there is more than one layer and you want to save the image visible in the Image Window to GIF, JPEG or some other format, you should first flatten the layers and then save the image.
- Saving an image with layers using the XCF format will save the image while preserving all the layer information.

The XCF format is the native file format for the GIMP and you should save all work in progress in this format. This format will necessarily produce very large files, much larger than the other file formats available. When you are done working on an image and ready to produce the final image into one of the many image file formats supported, you need to *merge all the visible layers* or

flatten the image. The latter process will remove transparency layers (after processing them) and constrain the final image to the visible size of the image. The former is similar, but preserves transparency and can be constrained to the actual image size, the size of the bottom layer or can be expanded as needed.

Flattening an image takes all the layers and combines them in a way that matches what you can see visually in the Image Window. Merging the layers can expand the image to encompass non-visible portions of some layers. In most cases, I've found flattening works better for my work. Merging can leave unwanted transparent regions that get translated to black or other colors in GIF or JPEG images. This doesn't mean there aren't any uses for merging, but initially, you'll probably find flattening works best.

There are actually two kinds of layers: normal and floating. A floating layer is a temporary layer that holds a pasted selection prior to applying it to either a new layer or anchoring it in the currently active layer. Floating layers have a few limitations due to their transient nature. Just be certain to convert the floating layer to either a new layer or anchor it by using the appropriate selection from the Layers Menu. Anchoring a floating layer causes it to be composited with the currently active layer or layer mask.

A layer mask is a tool for constraining which parts of a layer should be used in the composited image. This is best shown with a simple example. Let's start with a background layer with a simple gradient applied (Figure 4). Next, add a new layer (via the Layers Menu—click over the layer name with the left mouse button) and apply a different gradient. Then, add the layer mask by opening the Layers Menu and selecting Add Layer Mask. Make the layer mask fully transparent by selecting the Black toggle in the Add Mask Options dialog box. Make sure the new layer is the active layer by clicking on its preview image. The image state should look similar to Figure 5.

Figure 4. Background Layer Plus Gradient

Figure 5. Background Plus Two Gradient Layers

Now open a new image, a TIFF file that is a black and white image of a leaf (Figure 6). The leaf is white with a black background so that it will allow the current layer to show up in the interior of the leaf (the white areas) and allow the rest of the image to be transparent (allowing the first gradient to show through). Then select the entire leaf's image using “Select**->**By Color” and copy it to the clip buffer using the Image Windows menus (“Edit**->**Copy”). Paste that into the gradient layer and anchor it. The pasted image becomes part of the layers mask.

Figure 6. Leaf Image

Note that you can set whether a floating layer is anchored to the main layer image or to the layer mask by clicking on the preview images in the Layers and Channels dialog. The outline of the preview image is highlighted to show which part of the layer will be used to anchor the pasted image.

Now the image has two layers, with two different gradients applied. The top layers gradient only shows up where the leaf's mask is white. Figure 7 shows the final image with the Layers and Channels dialog.

Figure 7. Final Image

You can turn the layers mask on and off by holding down the control (ctrl) key while clicking the left mouse button in the masks preview image for the active layer. Alternatively, you can view just the mask by holding down the alt key and left-button clicking in the masks preview image. In the former case, visibility turned off creates a red border around the layers mask preview. In the latter case, visibility of the actual layer creates a green border. I don't use this feature much yet, but it's nice to know it exists.

The Layers Menu provides an Apply Layer Mask option which will composite the mask with the layer, if accepted. The same option provides the ability to discard the layers mask, in which case the mask is removed and no longer affects the layer or the composited image. You will need to apply the layer mask if you wish to use some filters or effects from the Image options in the Image Window menus.

A Quick Look At Filters

The GIMP is designed around the concept of external programs which handle much of the detailed work of the application, such as file I/O and image processing. The latter are commonly referred to as image filters since they filter the pixels in an image to produce an interesting effect.

To access the filters you use the Image Window's pop-down menu. The top level of this menu holds the Filters submenu. Under this submenu you will find an array of options including such categories as Artistic, Blur, Edge Detect, Render and many others. Each category has one or more filters associated with it.

Filters can operate on an image without any further user input, but most require the setting of variable parameters. When the filter is selected the GIMP launches the external program and opens a communications path to it. The plug-in filter then requests a dialog box with any number of buttons, menus,

sliders or other window features. Each of the configurable items in the dialog has a default value so the user can simply accept the defaults and have the filter process the image. Many of the filters provide preview windows to give the user an idea of how the image will look after the filter is applied based on the configuration the user has selected. Since some processing can potentially take quite a bit of time to complete, some filters allow the preview to be turned off.

It is impossible to discuss the breadth and detail of the available filters in this one article. I plan on writing some tips for using these filters in the future, either to be presented in the "Graphics Muse" column in the *Linux Gazette* or on my GIMP web pages. In the meantime, I'll just mention a few of my favorites.

The IFS Compose plug-in is a nifty little tool for creating fractal images. Fractals have been around for some time and many people have grown a little tired of the same old Mandelbrot sets, but this is different. The fractals are actually a set of three Serpinski triangles that can be rotated, scaled, stretched and colored. The transformation of each of these can, when done properly, create some very realistic looking foliage or very artistic backdrops. A nice feature of this plug-in is that it keeps track of its settings between invocations.

The Cubism and Mosaic Plug-Ins ("Filters**->**Artistic**->**Cubism" and "Filters**->**Artistic**->**Mosaic") offer some unusual effects. The Mosaic plug-in can produce images that appear as though they have been tiled using square and randomly edged 3D tiles. Cubism applies rectangular patches using the colors of the current layer and user-defined sizes and distributions. You can think of Cubism as "applying rectangular order" to a layer. Use of Cubism will cause a layer to be changed drastically so you might wish to duplicate a layer before applying this filter.

The Whirl and Pinch Filter ("Filters**->**Distorts**->**Whirl and Pinch") creates swirled patterns on the selected area or layer, much like having the image as the surface of a liquid in which you swirl your finger. The amount of swirl or pinch can be user defined, and a preview of the effect is available from the filters dialog window.

The SuperNova Filter ("Filters**->**Image**->**SuperNova") creates a bright sun-like section in the current layer. The Sun has spokes extending out from the center. The color and radius of the sun, the number of spokes, and the position of the sun within the layer are all user definable parameters for this filter. It is often useful to apply this filter on a transparent layer (with the "Keep Transparency" toggle turned off) and then add or subtract it from underlying layers.

And then...

There are lots of other filters, and the number of ways in which they can be used is endless. Next month I'll cover the Toolbox in its entirety, explaining what each icon is and how it is used. I'll also cover more details on creating and using selections as well as how to add and manipulate text within your images.

Resources

Michael J. Hammel (mjhammel@csn.net) a Computer Science graduate of Texas Tech University,is a software developer specializing in X/Motif living in Denver, Colorado. He writes the monthly Graphics Muse column in *Linux Gazette*, maintains the Linux Graphics mini-HOWTO, helps administer the Internet Ray Tracing Competition (http://irtc.org/) and coauthored the text *The Unix Web Server Book*, published by Ventana Press. His outside interests include running, basketball, Thai food, gardening and dogs.

Archive Index Issue Table of Contents

Advanced search

Ricochet Modem

Randy Bentson

Issue #45, January 1998

The reality is that these devices work quite nicely on desktop systems as well as on laptops.

- Manufacturer: Metricom, Inc.
- E-mail: info@ricochet.net
- URL: http://www.metricom.com/
- Price: $350 US
- Reviewer: Randy Bentson

First a confession, I didn't have to buy a laptop computer to use the Ricochet modem radio. The reality is that these devices work quite nicely on desktop systems as well as on laptops.

I read of these modems in the local free computer newspaper, but since I was quite happy with my 56Kbps frame-relay connection, I didn't give them much consideration. Then one day my connection failed and I realized I had no backup for Internet access. After a quick trip to a local computer store and a few minutes configuring a PPP connect script, I was back in action. The frame relay was fixed shortly thereafter, but I still have my Ricochet modem. I use it when visiting client sites and when testing firewall configurations, and I used it on the one occasion when the frame went down again. I've also loaned it to friends suffering from phone noise and ISP mismanagement.

How Does One Use It?

Plug it into your serial port. It has two connections: one for external power and one to connect to your computer. The computer sees it as a Hayes-compatible modem with a few additional AT (i.e., standard modem) commands.

I mentioned that I had to fiddle with the PPP connection scripts. As with most computer hardware, Metricom provides software for operation with the Apple and Microsoft operating systems. Since Linux comes with support for PPP dial up, there is no need to load their software. One has only to make some small but crucial changes to the scripts. The **ppp-on-dialer** script needs to send the strings:

```
ATZ
ATE0X4Q0V1&C1&D2&K3
```

to the modem to ensure that the configuration is correct. My **ppp-on** script needed the line:

```
route del default
```

because my system is also on a local LAN and normally gateways through the frame relay. The phone number, 777**PPP, looks unusual; it means “connect to the regular Internet service using the PPP protocol.” The script doesn't need a user name or password because the modem's serial number is used for authentication. As long as your account is current, your connection is established with a dynamically allocated IP address. Listings of these two scripts are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue45/2493.tgz.

How Does It Work?

First, the Ricochet is not a cell-phone modem. It is based on an infrastructure which is independent of the various telephone systems. Metricom has been in the wireless communication business since 1985—providing remote-access monitoring of meters to public utilities. The Ricochet division was started two years ago in order to offer wireless Internet connectivity to the public. They're currently offering service in the metropolitan areas of Seattle, San Francisco Bay, West Los Angeles and Washington, D.C. Metricom plans to extend coverage in the Los Angeles area by the end of this year and installation is underway in New York City.

If you pay attention, you'll see cell-phone antenna towers or roof-top clusters appearing everywhere. The Ricochet system is quite a bit harder to spot. There are three elements: the “modem radio” attached to your computer, a shoe box sized “microcell radio” attached to streetlamp poles at quarter mile intervals and “wired access points” to serve thirty or so microcell radios. (Seattle has 1800 microcell radios and 50 wired access points.) For instance, in this map of a Seattle neighborhood the small dots are the microcell radios, the red stars are wired access points, and the blue star is the home of *Linux Journal*. (The map is a product of a U.S. Census bureau server using the URL (without breaks) http://tiger.census.gov/cgi-bin/mapbrowser?

lat=47.676&lon=-122.366&wid=0.10&ht=0.050& on=GRID&murl=http://www.aa.net/~bentson/tms& iwd=640&iht=720. (See Figure 1.)

Figure 1. Seattle Area Map

Just as the Internet uses a store-and-forward model to get data from one place to another, Ricochet packets are forwarded from pole to pole from the modem radio to the wired access point, at which point they enter the conventional Internet routing. Because this service is packet based, you don't consume resources when you're not sending or receiving bits. Therefore, you're welcome to establish a connection and leave it live for as long as you wish. This is a dramatic change from telephone-based Internet services.

What's The Future?

Since I got my modem last fall, Metricom has offered two new products: a modem (without battery) for fixed stations and a newer, lighter weight version (8 ounce versus 13 ounce) of the portable modem. An even lighter version is being tested for palm-top computers. In addition, Ricochet has distributed a firmware upgrade that allows a roaming user to remain connected while moving from one wired-access-point coverage area to another.

The current technology operates in the 908-926MHz band, dividing it into 162 channels, each 160KHz wide. Each connection hops among these channels using spread spectrum technology. Modem radios provide a connection with performance comparable to a 28.8Kbps modem, but may burst higher. (That's why they recommend setting the serial port to 57.6Kbps.) Future models will use more channels and other bands to provide better coverage and higher speeds (128Kbps to 512Kbps).

What If You're Not In One of The "Blessed Cities"?

If you're traveling, you'll find service outside of these cities in airports such as Minneapolis-Saint Paul International Airport, Phoenix Sky Harbor Airport and La Guardia Airport in New York City.

Several schools, including Austin College, California Polytechnical Institute, George Washington University, Oregon State University, San Francisco State, Stanford University, UC Berkeley, UC Santa Cruz, University of Miami and University of Oregon, have established local service areas for their students, staff and faculty.

In addition to service provided in big cities, schools and airports, Metricom has teamed up with local utility companies to provide coverage in smaller towns—systems are being installed in Casper, Gillette, Laramie, Lander and Torrington

in Wyoming and Scottsbluff, Gering, Kearney, Alliance and McCook in Nebraska. The nature of the Ricochet system allows Metricom to offer localized service wherever there are enough users to support it—I have visions of service in places like Durango, Colorado.

Remember the strange phone number? You can also enter the serial number of another modem radio. If it's nearby and not currently engaged, it will see a "RING" and the other user's program can answer the phone. In this way a peer-to-peer service is established between two modems.

Figure 2. Map of Puget Sound Coverage

There's another form of operation as well—called Star mode. In normal mode, the modem acts like a Hayes-compatible modem; in Star mode, the modem delivers packets into space (the original ether) and other modems see these packets and deal with them in a manner similar to an Ethernet card. This mode only works in modem-to-modem communication; however, the greater bit capacity makes it worth exploring. If you look in the Linux kernel configuration, you'll find support for this mode is already available. With this code compiled in, you and your co-workers and friends can have your own wireless LAN wherever you go. It's very convenient for conferences.

About The Laptop…

I finally broke down and bought one. Naturally, it runs Linux, and when I'm out of the office, I unplug the PCMCIA Ethernet card and turn on the Ricochet modem.

When the microcell radios are installed, they're configured with their latitude and longitude. One of the extra **AT** commands causes the modem to report the location and signal strength of up to ten nearby microcell radios. I've wandered around Seattle getting these reports and looking for microcell radios. I edited these reports into the tms file cited in the URL above—that's how I built the map.

Needless to say, I had to check out the roaming feature once I got my laptop. As I drove around town, I issued a ping command directed at my home system. Sometimes the latency was quite high, but it also seemed to go down when a second ping was issued. I suspect that routing tables are updated often enough that the system quickly finds the best route. Now I've got to check it while riding the ferry across Puget Sound.

Visit http://www.ricochet.net/ for more information.

Randolph Bentson has been programming since 1969—writing more tasking kernels in assembly code than he wants to admit. He has been working with Unix for nearly 16 years, and he's been enjoying and contributing to Linux for the last 3 years. Mr. Bentson is the author of *Inside Linux, A Look at Operating System Development* [1996, SSC]. He can be reached via e-mail at bentson@grieg.seaslug.org.

Archive Index Issue Table of Contents

Advanced search

Red Hat CDE

Don Kuenz

Issue #45, January 1998

CDE provides a Graphical User Interface (GUI) to the X Window System.

- Manufacturer: Red Hat
- E-mail: info@redhat.com
- URL: http://www.redhat.com/
- Price: $79.95 US
- Reviewer: Don Kuenz

During the summer of 1997, Red Hat began shipping a version of the Open Group's Common Desktop Environment (CDE). CDE provides a Graphical User Interface (GUI) to the X Window System. As such, it shares some of the functionality offered by other window managers such as Feeble Virtual Window Manger (FVWM). A key difference between CDE and the others is that many of the leading Unix vendors provide a CDE package for their own native platform (AIX, Solaris, et al.) This means that users in a heterogeneous Unix environment need learn only one GUI, regardless of the underlying platform.

CDE's GUI is centered around a movable toolbar, which sits at the bottom of each of four workspaces. A workspace occupies one full screen and you can navigate between workspaces by clicking on a workspace panner, which is located in the center of the toolbar. CDE also features drag and drop functionality along with tear off menus.

What's Included?

Red Hat's Client Edition comes with a short Install booklet, a 300 page Advanced User's and System Administrator's Guide and a CD-ROM. The CD-ROM contains both software and lots of additional documentation. The User's Guide is well written with an elegance on par with Kernighan and Ritchie's *The C Programming Language*.

The Open Group bundles several new applications with CDE. Of these new applications, I use **dtterm**, **dtpad** and **dtfile** the most.

dtterm is meant to replace xterm and incorporates most of the functionality of xterm, adding a handy menu bar that allows you to cut, paste and set options. **dtpad** reminds me of Microsoft's Notepad. It uses **ctrl-c**, **ctrl-v** and **Ctrl-x** as hot keys to copy, paste and delete. If you spend any time using Windows, you should feel comfortable using dtpad. **dtfile** is a file manager that supports drag and drop as well as the normal options found in most file managers.

Installation

Running a simple script allowed me to perfectly install CDE over Red Hat Linux. But, CDE balked after I tried to slam-dunk (ignore all documentation, start the install script and press the enter key at each prompt) it over Slackware Linux. I'll take the blame for that failure, but it points out that you should carefully read the installation booklet if you install on a platform other than Red Hat.

Room for Improvement

In a CDE environment dtlogin replaces the command-line login. You enter your user ID with a password, and away you go—unless you are trying to login as root. In that case dtlogin denies the login. (See below for a workaround.) Red Hat needs to improve their install script to create a proper environment for root logins.

One other tiny improvement that Red Hat could make is to glue the User's Guide CD-ROM holder right side up. Somebody glued mine in upside-down.

The Open Group provides a generic login script, which causes some confusion in the Linux community. CDE sources a file named $HOME/.dtprofile to set up its environment. A comment at the bottom of .dtprofile implies that you can also source $HOME/.profile by setting **DTSOURCEPROFILE=true**, but that only works if you happen to use sh or csh as your shell. Unfortunately, most Linux distributions use bash, a shell that also sources $HOME/.profile.

Workaround for the root dtlogin Problem

The easiest way to handle this problem, especially on a stand-alone system, is to use Linux's virtual console. Most Linux distributions create four or more virtual consoles during the install process.

The only trick to using virtual consoles is knowing how to jump to them. To jump to the first virtual console, you press **alt-F1**. To jump to the second you press **alt-F2** and so on. **alt <-** or **alt ->** cycles you through all of the consoles and

displays, so to return to the CDE display you need to keep pressing one of those two combinations. With high resolution monitors, CDE requires a few seconds after you jump to it before it fires up. As you cycle through the screens, you can detect the CDE display because it's the only one that's blank—the remaining screens display a command-line prompt.

Conclusion

If you administer a heterogeneous Unix environment, it's a good bet that CDE can make your life easier by standardizing an X environment across all platforms. CDE also helps programmers develop and deploy cross-platform X applications with a minimum of fuss.

As a developer, CDE's four workspaces help me reduce clutter on my display. During a typical development session I'll open a terminal process in one workspace, use Netscape Navigator in a second workspace, use **ghostview** in a third and use **xxgdb** in the fourth.

I am very happy with my purchase. Red Hat delivers a solid product with good documentation that is well worth the purchase price of $79.95US plus shipping. Red Hat also sells a Developer's Edition for $199.00US. If you enjoy a nice GUI, you too will like CDE. CDE lends a lot of credibility to Linux as a production environment.

In 1975 **Don Kuenz** wrote his first program on a teletype machine connected to an HP computer, in BASIC. Don currently operates the software consulting company Kuenzsoftware, located in Casper, Wyoming. He also teaches computer science classes at Casper College. In his spare time he plays the piano, peddles around on a mountain bike and flies planes. He can be reached via e-mail at dkuenz@wind.cc.whencn.edu.

Archive Index Issue Table of Contents

Advanced search

Microway “Screamer 533”

Bradley Willson

Issue #45, January 1998

The idea of having that much “horsepower” behind the keyboard brought back memories of my 1970 Mustang and the feeling of being pressed into the seat back under acceleration.

- Manufacturer: Microway, Inc.
- E-Mail: info@microway.com
- URL: http://www.microway.com/
- Price: $2,995 US 500MHz and 433MHz available at lower prices.
- Reviewer: Bradley Willson

This story is proof positive that one has to be careful what they wish for, because the wish might come true. It all started when I joked with Marjorie Richardson of *Linux Journal* about reviewing a “Screamer 500”. The next item in my e-mail was a response stating that Carlie Fairchild had contacted Microway and had convinced them to send me a machine for review. My response was, “You're kidding, right?” Naturally, I was very pleased to find out it was true. The icing on the cake came when I called Ann Fried (pronounced “freed”) at Microway. She decided it would be better to have me review the new 533MHz machine instead of the 500MHz model. The idea of having that much “horsepower” behind the keyboard brought back memories of my 1970 Mustang and the feeling of being pressed into the seat back under acceleration. The only things missing were the roar of the engine and the smell of burnt rubber.

It is not the intent of this article to compare the Microway Screamer to other DEC Alpha-based computers. This article is about the profound differences between Intel 32bit Linux and Alpha 64bit Linux. I chose the Screamer because of Microway's advertisement in *Linux Journal*. Their ad provided the most information about the Alpha. Mainly this article is about having more fun with a

computer than should be legal, while getting a glimpse of the future of computing.

As a teenager, I thoroughly enjoyed racing against anyone that would pull alongside (State Highway Patrol was excluded, of course). Computers were not of interest to me then. Years passed, the hot rod was replaced by an econo-box commuter, and I discovered a new way to race. My first 286 was constantly tweaked to gain performance, then it was upgraded to a 386 and so on. Now the old 286 case sports an AMD 5x86 133. The green flag drops, the race between the AMD 133 and the Screamer 533 begins. My hot-rod Mustang was never a match for a Top-Fuel dragster, and the same is true for my computer pitted against a 533MHz DEC Alpha, but the fun part was getting to drive the dragster.

Let The Race Begin

Vrrrmmm, Vvvrrrrmmm, sputter, sputter, my office box fires up in DOS. Type linux at the C:\ prompt and Linux boots. I read 66.56 BogoMIPS from the scrolling screen. After I log in and enter **startx** I'm ready to go to work. Next I turn on the Screamer, wait about 4 minutes for the boot sequence to complete, read 1063.26 BogoMIPS from the startup messages. Immediately I can see the difference in speed. Parts of the boot messages are not readable because they scroll by so fast. I log into the Screamer and enter startx. The XFree86 configuration messages flash by so fast that they appear as a blur. Literally everything runs profoundly faster on the Screamer. Emacs appears within two seconds after menu selection. **xfig** is ready to use by the time I move my finger off the mouse button. Both of these applications take a few seconds to launch on my machine.

A Benchmark Exercise

I needed an application that would level the playing field and could be timed using “time” on both machines, so I wrote a bash script to append several instances of the phrase “The quick brown fox jumped over the lazy dog's back” to a file (see Listing 1). The task was repeated 100,000 times, creating a 5.2MB file on each machine. My 133 churned for 5 minutes and 12.98 seconds before completing the task (see Listing 2). The Screamer completed the execution in a mere 18.89 seconds (see Listing 3). I used the 4.9 minutes I got back from the Alpha to run a few more tests on it.

Even though it is a crude benchmark test, it exercised the CPU and the storage hardware of both machines. Several factors like IDE vs SCSI, memory and running daemons have an effect on the outcome, so your results will likely vary from these, but you will be able to gauge your machine's performance to the Screamer's.

Hardware Description

Microway builds an industrial-grade computer, starting with the case. The all-steel construction shows thoughtful engineering. Even the drive bay face-plates are made of steel. The side access panel makes working on the internals easy. Once inside, there is plenty of room to move around. Adding and removing storage devices is quick and easy thanks to the rail mount system. There is ample room between the drives and the motherboard and installed cards so you won't need a shoe-horn to install or remove a drive. The only problem I could foresee was that using the lowest two drive bays requires removal of the rivets that connect the card support rack to the drive support rack. Once done, it is left up to the user to provide the means to reconnect the racks. There are five fans to help the Screamer keep its cool while it crunches data. However, even with all those blades spinning, the Screamer is a quiet performer.

Like many other computers sold today, you can specify the options you want to buy with the base package, but the “basic” Screamer will likely keep any home user flying through code for years. The Screamer comes in two colors: black and beige. This particular box was built for the review and a series of trade-shows with these components: a 4.3GB Seagate Barracuda UW-SCSI hard-drive, a 1.44MB floppy drive, a 8x IDE Mitsumi CD-ROM, a Matrox Millennium card with 2MB, an Adaptec 2940 Ultra-Wide SCSI controller and 128MB of RAM. Of course, it also came with 2MB 9ns SRAM Cache installed. The motherboard sports 4 PCI slots (2x32bit and 2x64bit) and 2 ISA slots.

Microway generously sent a Iiyama 17'' Vision Master model MF-8617E monitor along with the Screamer. Using this monitor couldn't be easier. There are four buttons on the front panel and one of them is the power switch. You activate the on-screen menu with a single button and make menu selections with either the + or - buttons. I was impressed with its compact design. I've seen other monitors of the same display size taking up far more footprint on the desktop. This model has a maximum resolution of 1600 x 1200 non-interlaced, both 5-BNC and D-Sub mini 15 pin connections and a maximum of 19 user-definable signal settings. It is also power management and plug & play (VESA DDC1/2B) aware. At $679.00, it is a great value.

Software Description

Do not expect to find the same robust selection of applications and utilities for the Screamer as you would for Intel. The number is small in comparison to Intel-based availability. There are however, approximately 380 Alpha applications and utilities to be found at ftp://ftp.digital.com/pub/linux/redhat/redhat-4.2 and ftp://sunsite.unc.edu/pub/linux/ALPHA/alpha. Even with the relatively low number of available programs, there were several familiar applications installed on this machine, e.g., xfig, Emacs, mc, minicum and xfm.

Every Alpha program performed faster than the equivalent Intel program on my machine. The Screamer came with Red Hat 4.2 (Biltmore release) installed. I tried installing a number of the Alpha applications from my InfoMagic Developer's Resource set, circa September 1996. A few failed but the majority of them worked.

All is not lost when it comes to running Intel-based applications on the Alpha platform. Em86 (a non-supported product of DEC) will run numerous Intel-based applications. Em86 can be run standalone, in a script or as part of the kernel. It is best to apply the em86 patch and recompile the kernel, so that it can recognize the need to emulate when an Intel application is launched. Otherwise, you have to enter **em86** before every Intel application you wish to run. Furthermore, if the application spawns a child process, it will fail because em86 (standalone) won't spawn a process to perform emulation for the child process. With some planning and work, Intel ports can be used on the Alpha, but not without a performance penalty. Even so, using an emulator will go a long way toward maintaining operations while the Alpha ports are in development.

Installation

From the package truck to the office in 15 minutes—no speed limits were broken in the process. The precious cargo was handled carefully, as the phrase “you break it, you buy it” repeated in my head. The setup was like any other computer installation. I plugged in the keyboard, mouse, monitor and power cords. I flipped the switches and watched as the fun began. The only problem I had was making space for the full-size tower under my desk. The extra monitor on the desktop made for cramped quarters for a few weeks, but I certainly felt it was worth it.

Tech support

Microway offers telephone tech support at (508) 746-7341 and e-mail tech support at tech@microway.com and techsupport@microway.com. The sure-fire way to get answers is to call. I would like Microway to add an auto-responder to send back a response that tells the sender that their message got through and is in the works.

Company Profile

Microway has been producing advanced mathematical and scientific computing software products and hardware since 1982. Today they produce numerous NDP compilers in a variety of languages, i.e., C/C++, Fortran and Pascal, custom software and hardware. Their hardware product line reads like a who's who in industrial computing starting with Intel Pentium Workstations, i860 boards and,

of course, the Screamer series of DEC Alpha-based custom-built file servers and workstations. Companies like Ford Motor Company, General Motors Corporation, Rolls Royce and Fidelity Investments pepper Microway's customer database. Microway machines are installed at hundreds of universities worldwide.

Company Contact Information

Ann Fried was my sales contact. Mike Brown is the National Sales Manager. Nina Nitroy was very helpful by making a copy of **ldconfig** available on the ftp server. I had accidently overwritten the file with the em86 ldconfig. I couldn't shut down the machine without it. They all can be reached at 508-746-7341.

The Future

Microway will have released a new LX motherboard by the time this reaches the newsstands. It features six PCI slots (2x64bit and 4x32bit), two ISA slots and a 4MB cache. Microway LX motherboards are unique in the fact that they are backward compatible with slower but more affordable processors such as the 500MHz and 433MHz. Mike Brown of Microway pointed out that there is a five percent performance difference between the 500MHz and the 533MHz chips. Also look for their new NDP FORTRAN for Linux released back in October.

Bradley J. Willson currently designs and troubleshoots tooling for the Boeing 777 program and fills the chair of chief cook and bottle washer for Willson Consulting Services. His friends understand and forgive his addiction to computer technology, while others wonder how he can stand the countless hours he spends staring at screens. According to Bradley, the secret is attitude —and maybe a mild case of radiation sickness. He can reached via e-mail at cpu@ifixcomputers.com and http://www.ifixcomputers.com/.

Archive Index Issue Table of Contents

Advanced search

Running Linux

Zach Beane

Issue #45, January 1998

For those unfamiliar with Linux, and with little prior experience with any other Unix-type operating system, Running Linux offers a mountain of well-structured information to help get you on your feet.

- Authors: Matt Welsh and Lar Kaufman
- Publisher: O'Reilly and Associates
- E-mail: info@oreilly.com
- URL: http://www.oreilly.com/
- Price: $29.95 US
- ISBN: 1-56592-151-8
- Reviewer: Zach Beane

Running Linux provides a quick introduction to a wide variety of topics related to installing, configuring and using Linux. It covers installation in depth and introduces other topics by giving an overview with examples and referring the reader to other good resources. Depending on your view, the resource pointers, which are used extensively, can be a strength or weakness.

For those unfamiliar with Linux, and with little prior experience with any other Unix-type operating system, *Running Linux* offers a mountain of well-structured information to help get you on your feet. Unlike some other introductory

books, it rarely gets bogged down with too many details about a particular feature, task or program.

In the course of reading *Running Linux*, you will be introduced to virtually all aspects of using and maintaining your new Linux system. For those left puzzled by their first root shell prompt, it provides a comprehensive path to becoming a Linux user/administrator, from adding accounts to changing passwords to configuring printers.

From there, *Running Linux* also provides information about fun, interesting and even profitable things to do with your system. Each of these sections are laid out clearly and conveniently. The chapter on programming, for example, offers an overview of popular editors, debuggers and their interfaces and programming languages, giving you a taste of each and allowing you to follow-up with other references.

The references, too, are comprehensive, including HOWTOs, info and man pages, web and ftp sites and other books. Readers may be disappointed at the frequent need to refer to outside sources to get more information—this definitely isn't the only Linux book or resource you'll ever need. From this shortcoming, however, comes an interesting strength; the book often gives pointers to resources that are continually kept up to date, eliminating the pitfall of going into depth about a program feature and then having a new version of the software render the information obsolete.

If you're an experienced Unix or Linux user, you may find that there isn't much material covered that you haven't run into at one time or another. Since no subject is covered in great depth, you're not likely to find some hidden piece of information about a program you've been using for a while. However, the scope of the book is such that even if you've been hacking away for a few years, you'll probably still find some bit of information you never knew before. In addition, if you haven't used GNU or other freely available tools before, *Running Linux* is a fairly good introduction, although if you have a specific tool in mind, a specific book would probably be a better source of information.

While *Running Linux* is in its second edition, there are a few fairly dated passages, but these are not common. This is quite forgiveable, for Linux is constantly changing and updating, and any book about Linux initially written over two years ago is bound to have them. Another minor disappointment is the chapter on multimedia which provides only a cursory glimpse at the subject.

Running Linux is a worthwhile book for the Linux newcomer. Veterans to Linux and Unix may find it interesting, but not as useful.

Zach Beane is a database/web guy, Perl programmer and assistant system administrator at The Maine InternetWorks, Inc. In his spare time he likes to play Linux Quake and is slowly learning the rocket jump. He can be reached at zach@mint.net.

Archive Index Issue Table of Contents

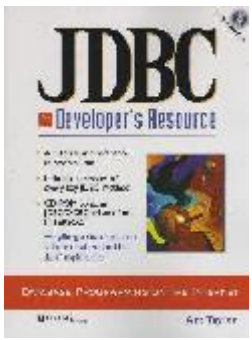
Advanced search

JDBC Developer's Resource

Rob Wehrli

Issue #45, January 1998

This Art Taylor book, published by Informix Press/Prentice Hall, is rich with content and worthy of more careful study by anyone but the most astute Java professional working with JDBC.



- Author: Art Taylor
- Publisher: Prentice Hall, Inc.
- URL: http://www.prenhall.com/ — developers_resource_series/
- Price: $49.95 US
- ISBN: 0-13-842352-0
- Reviewer: Rob Wehrli

Java is hot. Java with Java DataBase Connectivity (JDBC) is very hot. This Art Taylor book, published by Informix Press/Prentice Hall, is rich with content and worthy of more careful study by anyone but the most astute Java professional working with JDBC.

True to its cover, Taylor's book features:

- A tutorial and reference in one volume
- Examples of every key JDBC method
- CD-ROM containing JDBC/ODBC drivers from Intersolv

JDBC Developer's Resource also comes very close to being “Everything a developer needs to build database-enabled Java applications.” Not reported on the cover, but inside, are examples of two and three-tiered applications. The JDBC tutorial offers a hands-on, step-by-step approach to implementing the power of JDBC, that is not for those afraid of getting their hands dirty in properly commented code. The one buzzword missing from the cover is “practical”--it is a practical book, and its “Quick Reference” enforces that impression.

Generally speaking, I like the book. In fact, I was asked to review it, because I am very involved in database design and am a systems and software engineer. Currently, about 90% of the code I write is for server-side Java applications for the Internet. About 99% of these applications use some kind of database back end. Of these, I'd guess that 75-85% use the JDBC-ODBC Bridge. While these relational database management systems (RDBMS) may not be as exciting to me as object-oriented database management systems (ODBMS) with native Java bindings, they are by far the largest contingent of installed systems. It is this portion of the database world addressed by JDBC.

The content of this book addresses the many issues surrounding implementation of JDBC in Java applications by providing many excellent, working examples through a logically structured presentation of information. It starts with basics and develops into a number of higher level concepts. It even has a brief SQL introduction for newcomers to RDBMS. I am not overly fond of the graphical layout of this manuscript, but I consider layout to be a matter of personal taste rather than a design flaw. Many code samples feature black ink on a rather dark gray background. The included CD-ROM features a number of cool goodies including a collection of drivers from Intersolv.

In a classroom grading scale, I'd give it a B+ for content and structure, and a C+ for presentation. I'd give the JDBC quick reference guide an A-. Would I buy this book? Based on the content, I'd certainly buy it, but while shopping for a book, I'd look at other books on the shelf for matching content with a better visual appeal to me. If you're new to JDBC and want a quality tutorial, buy this book.

Resources

Rob Wehrli is a systems engineer and longtime resident of Honolulu, Hawaii. He enjoys playing golf and chess. He can be reached via e-mail at rowehrli@pixi.com.

Archive Index Issue Table of Contents

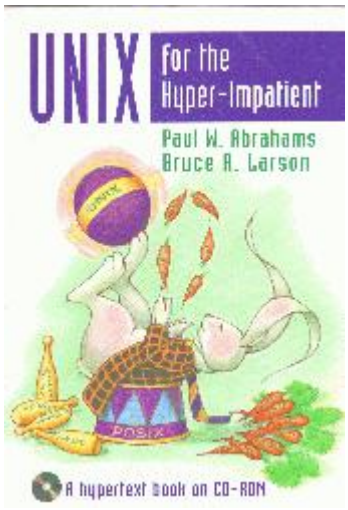
Advanced search

UNIX for the Hyper-Impatient (CD-ROM only version)

Daniel Lazenby

Issue #45, January 1998

The book is actually a hypertext repackaging of UNIX for the Impatient, Second Edition.



- Authors: Paul W. Abrahams and Bruce R. Larson
- Publisher: Addison-Wesley
- URL: http://www.awl.com/
- Price: $28.95 US for the book or the CD-ROM and $49.95 US for both
- ISBN: 0-201-41991-2
- Reviewer: Daniel Lazenby

Looking for an on-line reference that cuts to the chase and is written toward the technically oriented and somewhat Unix aware? If so, *UNIX for the Hyper-Impatient* may be worth your consideration.

The book is actually a hypertext repackaging of *UNIX for the Impatient*, Second Edition. The content of the original book contained many internal cross references and pointers to related information. I believe the existence of these “links” is one of the reasons the book was released in an all-electronic hypertext

format. There seems to be little difference between the content of the electronic and hard copy versions of the book.

The book is organized functionally and divided into fourteen chapters and six appendices. You might want to read the Introduction and Concept chapters sequentially. Then again, you may find yourself randomly skipping in and out of the other chapters. Chapters 3, 4 and 5 are functionally arranged to answer the question “What command(s) can I use to ___” (you fill in the blank). Chapter 3 focuses on operations that may be performed on files. Chapter 4 addresses data manipulation using filter commands. Chapter 5 discusses utility programs used to monitor and manage your Unix environment. Titles of the remaining chapters include Chapter 6 “The KORN and POSIX Shells”, Chapter 7 “Other Shells”, Chapter 8 “Standard Editors”, Chapter 9 “The GNU Emacs Editor”, Chapter 10 “Emacs Utilities”, Chapter 11 “Mailers and Newsreaders”, Chapter 12 “Communicating with Remote Computers”, Chapter 13 “The X Window System” and Chapter 14 “Managing Your System”. The Appendix contains an Alphabetical Summary of Commands, Comparison of MS-DOS and Unix, a Resource list, a Glossary and an Index.

Chapters 6 through 14 are largely factual in nature. These chapters often state the basic facts, concepts and identify the relevant files. To me, these chapters seem to present the “what”, “where” and a little of the “why” of the selected topic. An example of this approach is the X chapter, which opens with an explanation of what the X Window System is and what it does for you. It continues by describing how the various parts of X look and what they do. This chapter then describes the high level flow of events that initiate an X session, including the role the X initialization files play in the initialization process. Descriptions often contain information as to why one would be interested in the particular X file, or would want to use the particular X feature or function being discussed. This chapter on X is a complete walk through the X environment, files and file contents. From a user's perspective, few stones have been left unturned. This type of reference would have saved me a lot of angst when I encountered my first X environment.

The Alphabetical Summary of Commands is divided into an alphabetical listing and a summary of the commands. There are two hypertext links for each command in the alphabetical listing. One link jumps to a summary of the command and the other jumps to a detailed explanation of the command. The command summary provides the level of detail I associate with O'Reilly's *Linux* (or *Unix*) *in a Nutshell* book. Detailed command explanations are well written descriptions that include examples.

Addison-Wesley's web site provides access to the preface of this book. Reading the on-line preface will give you a feel for reading an entire book on-line. It is also a good example of the authors' style.

The Hypertext Browser

A product called DynaText displays the hypertext book and provides the user interface (see Figure 1, Reader Window). In addition to the expected read, browse, search and print functions, DynaText provides the user with the ability to define his own bookmarks, notes, and cross references. A journal feature allows the user to record and retrace his path through a series of topics.

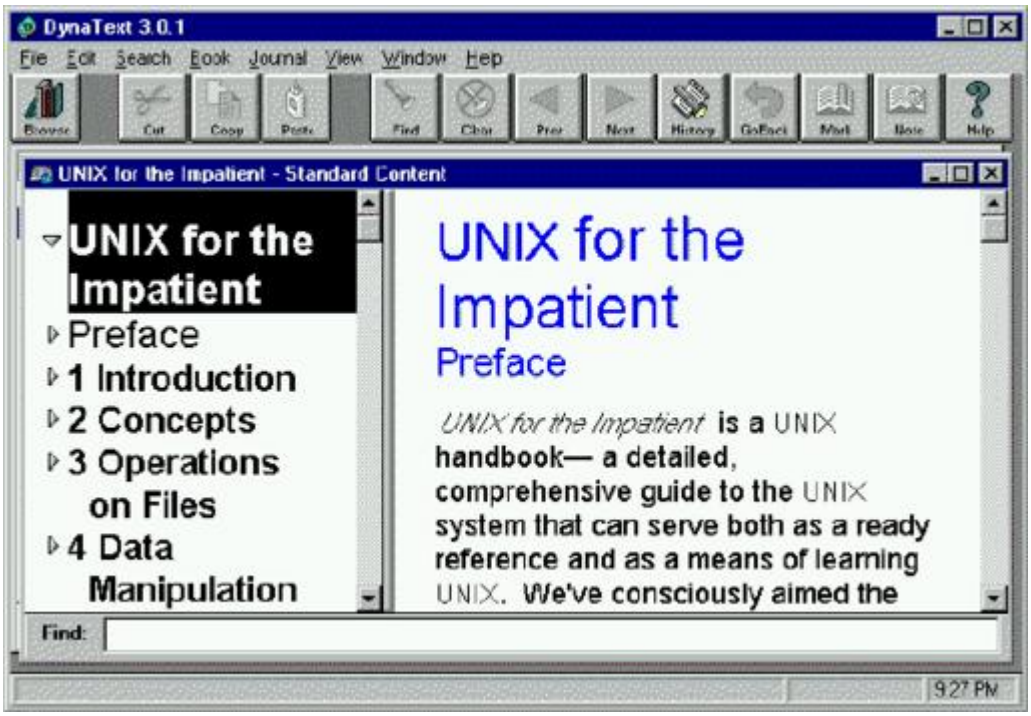


Figure 1. Reader Window

About ten short pages describe how to use the DynaText software and supply the meaning of the various icons. The DynaText on-line User's Guide is also useful, as some of the interface features work a bit differently than what you might expect. For example, the “Bookmark” button remains greyed out—until you actually highlight some text. Clicking on the active Bookmark button produces a dialog box to record your bookmark. The button greys out again after the bookmark is defined. The same process is true for the “Notes” button. User-specified names are assigned to bookmarks and notes at the time of creation. DynaText did not allow me to use the same name for a bookmark and a note I associated with it. Buttons on the tool bar are nice, yet I got more utility out of the bookmark and notes accelerator keys. The appropriate “Annotation Manager” menu option must be used to access any user-created bookmarks or notes.

Most of the hypertext links are colored green and quite obvious. These green hypertext links may be a title, a Chapter section number, a page number or an Appendix reference number. Another set of hypertext links is indicated with one of several icons. A third type of hypertext link presents no immediate visual indication of its existence. If a pointing finger cursor appears when you place the cursor over a word, then there is an active hypertext link. This style of a hypertext link is used with glossary words. Clicking on any one of these hypertext links opens another window containing the referenced content.

Every predefined hypertext link opens a new window. This technique makes it easy to see text at both the point of origin and destination. On the other hand, this technique can quickly cause the screen to become cluttered with windows. With several windows open, I found it difficult for me to tell which window belonged to which hyperlink jump.

Considering the keyword search capability of the browser, I was disappointed not to find an accelerator key that placed the cursor into the on-screen search field. I found one other search characteristic annoying. The browser left the search word in the search field upon completion of the search. Since each jump opens a new window, as soon as a new window opened, the browser would search for the word in the previous window's search field. I received several “not found” dialog boxes that I had to clear after making a jump.

A VCR style button panel is used to represent the “Journal”. A Journal is created by clicking on the “Record” button and navigating through the book. There is no need to worry about wrong turns or jumps. The Copy, Paste and Cut buttons allow you to re-sequence or remove topics. You can even add topics you may have missed. Once established, this journal may then be used to retrace the same path or to guide another person through the same set of topics. There are two journal playback modes: continuous and frame by frame. Out of the box, the continuous playback interval is set to three seconds, but can be adjusted. Frame by frame uses the VCR fast forward and reverse buttons to page forward or page backward, one page at a time. The VCR panel is a bit large for my tastes. I found myself moving it around the screen to get it out of the way while recording and during the playback.

If you do not feel the predefined links meet your needs, you can add your own one- or two-way hyperlink to the book. A hyperlink simply jumps from one book location to another. Creating a user hyperlink is as simple as marking text at the starting point, the ending point, indicating one- or two-way link and naming the link. A user-created hyperlink does not open a new window. This type of link jumps directly, within the same window, to the destination point. As with Bookmarks and Notes, you may have to use the Annotation Manager to locate user-created hyperlinks.

Printing excerpts produced some odd results. I selected two short chapter topics to print (Chapter 6.5 and 13.1). Each of these sections happens to have a "Page Icon" on it. These sections were printed across two pages. The first few lines of the section were printed on one page and the remainder was printed on another page. Several words were dropped from the sentence split between the two pages. Selections and chapter sections did not contain a Page Icon printed as expected.

Resources

Where does one go for help?

I had a Win95 installation glitch that did not make sense. The hypertext links would not display properly. A large majority of the hypertext links kept appearing as a green "fred." I repeated the installation a couple of times to make sure I wasn't missing anything in the instructions. (Yes, I really do read them.) After a couple of reinstalls the glitch remained, and it was time to seek assistance.

The book's *Installation and User's Guide* published the e-mail address help@qsep.com for installation assistance along with a voice and a FAX number. A Web search for Quickscan produced a web site hit (http://www.qsep.com/). I was unable to reach any of these choices. I later learned that QSEP was having major difficulties with their web server and support line at the time I was reviewing this product. Prior to making contact with QSEP I had stumbled my way through the hyperlink display glitch by installing the program to run directly from the CD-ROM. It seems a couple of other people encountered a similar symptom. Not having had a chance to work through this problem directly with QSEP, I am unable to comment on their quality or level of support.

Buried in Addison-Wesley's list of titles for Mr. Abraham's web page is a URL pointing to browser update information (http://www.qsep.com/unixbook.htm). Other than this URL, I was unable to locate any form of software support for this specific title on the Addison-Wesley web site. I did locate an "Ask/Tell Us" web page that provided the means for sending Addison-Wesley a comment and the ability to indicate the comment was technical support related. While Addison-Wesley maintains a link to QSEP, I would not count on Addison-Wesley for *UNIX for the Hyper-Impatient* technical support.

Inso Corporation's technical support for DynaText is password protected and specifically directed toward individuals who have purchased the DynaText product directly from Inso, or have purchased a maintenance contract. Inso clearly states that persons who are seeking DynaText support and did not buy it

directly from Inso need to contact the “third party” vendor who sold them the DynaText-based product. I was also informed that any of the updated browsers would have to come from the “third party” vendor as well. So it seemed one can't even get a copy of the browser's current release from Inso. In spite of Inso's “We didn't sell you the product so don't ask us for the latest browser policy,” Inso was very helpful. They provided prompt responses to any question regarding DynaText product plans and Unix OS compatibility.

Supported Operating Systems

Conclusion

I liked the book's no nonsense approach and content. Each chapter seems to be focused on “What do I need to know to do this task.” Whenever I read a book, I'm always making notes and inserting cross references and notes of my own. This book is rich with internal cross references and footnotes.

I'm not really sure about the DynaText browser though. The lack of a supported Linux browser for this review was disappointing. I did have limited access to an AIX 4.1 client machine. That avenue of review was a dead end as well, since the browser for AIX 4.1 was still in the works. I would have liked to see how the DynaText product behaved in a supported X or Common Desktop Environment. Before purchasing this CD-ROM, I'd suggest checking the above sites to verify that there is an available DynaText browser for your flavor and version of Unix. Downloading a browser could be a lengthy endeavor. One browser reference indicated it was close to 17MB.

Daniel Lazenby lives in Arlington, Virginia. For a living he supports and works with AIX and RS/6000 systems. In early 1995 he discovered Linux and began using and tinkering with it in the evenings. He may be reached at dlazenby@ix.netcom.com.

Archive Index Issue Table of Contents

Advanced search

Internet Connections With the 56Kbps Modems

Tony Williamitis

Issue #45, January 1998

Higher speed Internet connections are on the horizon with U.S. Robotics' XS modem and Rockwell International's K56Plus.

Most of us are deeply concerned about the speed of our Internet connections. As we browse ever more complex and graphical pages on the World Wide Web, we often have time to read the newspaper while waiting for the desired information to appear on our screens. And we have all heard that our V.34 modems, which can communicate at up to 33.6 thousand bits per second (Kbps) are the final word in speed over our normal phone lines. The technological advances which used to double our connection speeds every few years have finally come to an end. So why are major modem players like U.S. Robotics and Rockwell International announcing new modems which can communicate at up to 56Kbps?

A major shift in how the Public Switched Telephone Network (PSTN) is constructed has taken place and has opened the door to this latest increase in speed. For over 100 years the telephone system was an incredibly complex analog switching system of circuits connecting telephones to other telephones. Recently, the PSTN was substantially upgraded to digital high-speed connections, often carried over fiber-optic cables. This modern telephone infrastructure can now be utilized for ever-higher data speeds.

Also, a new method of data encoding has been developed and will soon be available. This technology is called X2 by U.S. Robotics and K56Plus by Rockwell International. The systems are similar enough to be considered the same for the purpose of this discussion, and I refer to both as 56Kbps technology. It remains to be seen whether the implementations chosen by these two companies will be compatible.

In order to explore this new technology, a little background information on the public telephone system is in order. When you place a call to another person

you probably don't think about how the connection is made. The telephone system is so reliable and easy to use that any 5-year-old can easily place and receive calls. Behind the scenes, however, is a very complex system of electronic equipment which is not widely understood. Most connections today are handled by digital switching equipment while almost all residential telephones are still analog. In fact, your residential line has changed little in the past 70 years. The changes have occurred inside and between the Central Offices (COs). Normal modems, including the latest 33.6Kbps models, assume they are communicating over a connection that is completely analog. This assumption is no longer always valid. There are still many analog switches in use in the United States, especially in rural areas, but they are being quickly upgraded to digital systems.

Your phone line is basically two copper wires which carry current between your telephone and your CO. Your voice is inherently analog, which means it is continuously variable and contains an infinite number of discrete levels. Your telephone converts your voice to fluctuations in the current carried over your phone line. Upon receipt at the CO, however, a major change occurs. Your phone line is connected to a piece of equipment which converts analog signals to digital and digital signals to analog. This equipment is called a *codec*, which stands for COder/DECoder. After your voice has been converted to a digital signal, it can be processed by the digital switching matrix in the CO and transported over digital lines to another CO or to another line in the same CO. Another codec converts this signal back to analog fluctuations in current on the copper wires which lead to another telephone. The other telephone then converts this current into an audio signal which can be understood by a listener. A similar process occurs going the other way, and you hear the other person's voice, no matter how far away from you he may be.

This digital-to-analog and analog-to-digital conversion is the area of interest to the designers of this new modem technology. A coding standard has evolved in the United States, and similar standards are in place in Europe and other parts of the world. The analog signal is sampled 8000 times per second by the codec and each sample generates an 8-bit byte of data, which can take on 256 distinct values from 0 to 255. These samples are taken continuously, resulting in a stream of data which is 8,000 X 8 = 64,000 bits per second or 64Kbps. It is this data stream which actually gets transported between COs and is usually combined (“multiplexed”) into another, higher-speed, connection. For example, 24 of these 64Kbps streams can be combined into one T1 line, which results in 1.544 million bits per second (Mbps). This rate is greater than 24 X 64,000 because additional bits are used for timing.

It is important to note that sampling is inherently imperfect. The codec attempts to quantify an infinitely variable signal into 256 discrete levels. Each of

these levels is necessarily a best estimate of the actual value. The error introduced is called quantization noise and is measured in relation to the desired signal in decibels (dB). This is known as a signal-to-noise ratio. It is important to note that the corresponding conversion of digital to analog does not cause quantization noise since no error occurs: each of the 256 levels is correctly translated to a distinct voltage.

Because the telephone system was designed for voice, it was proper engineering practice to tailor the circuits to the characteristics of the human voice. Almost all of the energy in a human voice is contained in a band of frequencies from 300 Hertz to 3300 Hertz. (Hertz signifies cycles per second and is a measure of frequency or pitch.) Thus, the telephone circuits have been limited by design to this range of frequencies. This 3,000 Hertz range is known at the *bandwidth* of the telephone connection. It is this bandwidth which limits the speed at which data can be transferred.

The recent development of V.34 modems makes maximum use of this bandwidth and approaches the theoretical maximum speed of 33.6Kbps. This limit is due to quantization noise as discussed above, and is determined by Shannon's Law. Shannon's Law defines the maximum data rate over a connection as a function of the connection's signal-to-noise ratio. This calculation is beyond the scope of this discussion but results in a theoretical upper limit of approximately 35Kbps for an analog phone line.

Now that we understand a little about the phone system, let's look at what happens to data as it travels from your computer to a local Bulletin Board System (BBS). Your data is born digital in your computer and translated to analog by your modem. At the phone company's Central Office, the analog signal is sampled and converted to a digital stream. This digital stream is switched and transported to the CO serving the BBS, where it is converted to analog for the local loop serving the BBS. At the BBS modem, the data is finally converted from analog to digital one last time, and it can now be understood by the BBS computer. If you were following closely, you noticed four separate conversions, two analog-to-digital and two digital-to-analog. Remember, when an analog-to-digital conversion takes place, quantization noise is introduced.

Now we start to understand why our bandwidth is limited. Recent technologies, such as ISDN (Integrated Services Digital Network), can bring 64Kbps, or even 128Kbps, to our computers. But ISDN requires expensive upgrades at the CO, and, therefore, much higher monthly charges when compared to normal analog phone service. 56Kbps technology requires no change to an already digital CO.

The good news, and the enabling factor for the new 56Kbps technology, is that most large service providers now use modems which skip two of the above-mentioned conversions. Because these service providers connect to the Central Office by way of digital lines such as T1s (1.5Mbps) or T3s (45Mbps) and ISDN, digital modems can be employed which do not need to translate data for transport over an analog local loop from the service provider to the CO. Here lies the secret behind 56Kbps technology. Recall that quantization noise results from an analog-to-digital conversion. If the service provider can avoid this conversion, the noise is not introduced. Theoretically, if this conversion step is eliminated, the entire 64Kbps data channel cad be exploited.

However, in the real world, only 128 of the possible 256 voltage levels will be used, resulting in 56Kbps transmission from the service provider to the home. Notice that the analog to digital conversion step from the home to the service provider has not been eliminated and, therefore, no increase in speed is afforded in that (upstream) direction, so standard V.34 33.6Kbps transmission continues to be employed. This indicates that the 56Kbps technology is asymmetrical in nature, resulting in different speeds in different directions. Also note, 56Kbps technology will only work when connecting to another modem which is served by a fully digital connection—you will not be able to enjoy 56Kbps transfer speed when calling your friend or a small bulletin board system. Normal data compression techniques will still be used as they are today to increase throughput. For example, a text file transmitted over a 56Kbps data link using V.42bis compression will result in a transfer speed of 230.4Kbps. That is quite an improvement over today's available transfer rates.

To wrap all this up, let's note a few important facts. A service provider must have a fully digital connection to the Internet, either T-carrier or ISDN. The service provider must have 56Kbps modems which are compatible with your modem. U.S. Robotics announced the availability of X2 upgrades in January 1997, and other modem manufacturers are expected to make upgrades available sometime in the first quarter. Some modems will require only software upgrades, while other modems will require hardware changes or will not be upgradable at all. In general, a late-model modem will be software upgradable if it uses FLASH ROM (the on-board firmware can be upgraded without physical replacement of ROM chips). As noted earlier, it is far from certain that the different manufacturers' implementations of 56Kbps will be compatible, so care must be taken to ensure your modem is using the same encoding scheme as your service provider's modems.

Further information on 56Kbps technology can be obtained on the U.S. Robotics web page at http://x2.usr.com/ and on the Rockwell International web page at http://www.nb.rockwell.com/mcd/K56Plus/.

Tony Williamitis can be contacted via e-mail at twilliam@eos.net. Your feedback concerning this article is welcome. Links to information on various telecommunications subjects can be found on the author's web page at http://www.dma.org/~twilliam/.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

A Recipe for Making Cookies

Reuven M. Lerner

Issue #45, January 1998

Cookies are an excellent way of keeping track of users who visit a web site. Here's how to use them.

The overwhelming majority of URLs begin with the letters “http”, which stands for “hypertext transfer protocol”. Just as e-mail is transferred using SMTP (Simple Mail Transfer Protocol) and files are often retrieved using FTP (File Transfer Protocol), files written in HTML are generally transmitted using HTTP.

Why did the Web's inventors create a new protocol for transmitting hypertext, rather than sticking with previous ones? One answer is that they were interested in allowing servers to respond quickly and efficiently to requests from browsers. The client (browser) side of an HTTP transaction consists of a request for a document, containing several optional parameters, describing the document's content type and its last modification date. The server responds to the request by describing the document, including its content type, and returning the document. Once the document is sent, the server closes the connection. By exchanging a minimum of information and then breaking the connection, documents are transmitted with a low overhead, and thus, at a relatively fast clip.

This “statelessness”--the fact that each connection is used to transmit a single document and that each transaction takes place in a vacuum—was a terrific idea in the early days of the Web. It meant that browsers and servers had to keep track of very little information when transmitting documents, thus reducing the size and increasing the speed for these programs.

As a result, if we look at the access log from a typical web server, we see a list of document requests as well as the IP address (i.e., the number that uniquely identifies a computer on the Internet) of the computer from which the request originated. We do not, however, know whether three requests made from the

same computer at roughly the same time were made by the same person or by three different people.

In many cases this would not be a problem; after all, if my web site is set up to serve out pages of HTML, then I probably don't care whether 1,000 different people have visited my site or if the same person has read 1,000 documents. For many sites statelessness does not present any obstacles.

However, many site owners, particularly commercial ones, are increasingly frustrated with the Web's inherent statelessness. It is much easier to sell advertising when you have a precise count of the number of people visiting your site, rather than a list of how many times each document was accessed. The number of “hits”, or individual HTTP requests received by a server, is a reasonable measure of a site's success only in the non-profit and personal sector; commercial sites are far more interested in how many pages were viewed by a given number of individuals.

Even small personal sites occasionally like to keep track of users. If you want to personalize a user's view of your site, a way to keep track of each user's preferences rather than a setting which applies to all users. And, while you could certainly get a user's name (and password, if necessary) via HTML forms, forcing the user to enter this on every page, or even upon arriving at your site's home page, would be a great burden on the user.

This month we will look at one of the most popular ways to keep track of user state, best known as HTTP cookies. Cookies allow servers to store small pieces of data on the user's computer, and thus to keep track of a user's movements on our site. Note that while cookies can be used to keep track of a user's movements, and potentially build a profile which might be of use to advertisers, they cannot collect any information which the user does not provide. Fears of privacy abuse might be true in some cases (and designers should recognize that cookies will offend and upset some users), but the fear that cookies can somehow collect information from your computer without your knowledge is off the mark. Cookies simply make it much easier to create interesting sites.

What is a Cookie?

Cookies are small (up to 4KB) pieces of data stored on the user's computer by his browser. In addition to a name,value pair, cookies are tagged with expiration dates limiting the length of time they may be stored, as well as an indicator of the Internet host or domain that originally created the cookie.

The basic rule to remember when dealing with cookies is that the value of a cookie is set by the server using HTTP responses, and browsers return those

values using HTTP requests. It's a bit disconcerting to think of things this way; we are not used to responses from servers containing a request of their own.

Let's say that we have a CGI program that returns a small bit of HTML when invoked. Assuming that the program is in the /cgi-bin directory and is called sample.pl, our browser would retrieve it by connecting to the server on port 80 and issuing a request like this one:

```
GET /cgi-bin/sample.pl HTTP/1.0
```

This request says that we are using HTTP 1.0 and would like the server to send us the document /cgi-bin/sample.pl. The server, because of its configuration options, knows that anything in /cgi-bin is a program, and so it executes sample.pl, returning the output. Here is an example of what sample.pl might return:

```
HTTP/1.0 200 OK
Content-type: text/html
  <HTML>
  <Head><Title>Test</Title></Head>
  <Body><P>Test</P></Body>
  </HTML>
```

The above is about as minimal as a modern HTTP transaction can get. A single header (**Content-type**) following the status code and preceding the message body is returned. Most of the time, more information is included in the response headers, such as the server name and version number and the date on which the document was created. If the server wants to set a cookie on the browser's computer, it must include an additional header, named **Set-cookie**. Just as the **Content-type** header defines the type of data that is being returned in the response, the **Set-cookie** header defines the name and value for a cookie that applies to the site from which the response originated.

For example, Listing 1 contains a short program (cookie-test.pl) that creates a cookie on the user's computer. If we run cookie-test.pl from a web browser, we see the HTML output produced by the program. If it were not for the program's polite indication that it had set a cookie, we would never know unless we asked our browser to warn us each time. (I tried this feature on discovering it in Netscape Navigator 3.0, but I quickly turned it off when I discovered how often such dialog boxes were interfering with my web browsing and how innocuous most of them appeared to be.)

How Browsers See Cookies

The **Set-cookie** header becomes obvious if we use **telnet** to look at the output sent by the program. From my computer running Red Hat Linux 4.2, I type:

```
telnet localhost 80
```

which opens a connection to the Apache HTTP server running on my computer. I then type:

```
GET /cgi-bin/cookie-test.pl HTTP/1.0
```

followed by two line-feed characters, which indicates the end of my request. As in the example above, my server knows that anything in /cgi-bin is actually supposed to execute cookie-test.pl and to send the output from that program to the user's browser. When I enter the above request and press the return key twice (once to end the request line and another to indicate that we have finished the entire request), I get the following:

```
HTTP/1.1 200 OK
  Date: Tue, 23 Sep 1997 09:15:42 GMT
  Server: Apache/1.2.4
  Set-cookie: counter=1; path=/cgi-bin/
  Connection: close
  Content-Type: text/html
  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
  <HTML><HEAD><TITLE>Cookie set</TITLE>
  </HEAD><BODY><P>The cookie named "counter" has
  been set to 1.</P>
  </BODY></HTML>Connection closed by foreign host.
```

The above response is a bit more detailed than the skeleton response that we looked at above, but its contents should still be fairly clear. We get a 200 (“OK”) message from the web server, the date at which the document was created, the server that produced the response, the connection type and the content type.

The **Set-cookie** header tells our browser that we should now hold onto a cookie named *counter*, whose value should be 1. In the future, every time my browser asks for a document in the cgi-bin path from this host, it will send the value of the counter cookie, which will still be set to 1. Listing 2 is a short CGI program that prints the names and values of all cookies sent to it.

Note that our program only sees a single cookie, while I know that my browser has held onto far more cookies that this. I do not have to enter my password when entering certain sites at which I have registered, and there are a large number of cookies in ~/.netscape/cookies, the file in which the Netscape's browsers place cookies. Why does only one cookie appear?

The answer is that when a browser visits a site, it only sends those cookies that were created by that site. Thus, when I am accessing my local web server, only those cookies created by my local web server are available to CGI programs there. If I were to access *The New York Times*, only those cookies set by the nytimes.com domain would be available to their system. One of the cornerstones of the cookie specification is that the cookie name and value pair should only be sent to the hosts or domains that created them.

Those of you worried that information about your web browsing interests is shared among sites (and thus violate your privacy) might still be right—but this cannot happen automatically with cookies unless all of the sites you visit are within the same domain. Indeed, HotWired used to have several sites with a shared password system that relied on cookies. Because the sites all had different domain names, however, I had to have a separate cookie on my system for each sub-site. The result was that I had to enter my user name and password the first time I visited each of these sites.

Storing Useful Values

Now we know how to create cookies. Our CGI program uses CGI.pm's **cookie** method to create one with a name and value, and then puts it into the header returned to the browser. We also know how to write programs that can grab cookies' values. To get a list of all cookies, use the same **cookie** method, then iterate through the list of names that it returns. Once we have those names, we can retrieve the values with the **cookie** method, giving it an argument of a particular cookie name.

Storing a value isn't of much use unless we can also change it. Our next task is to combine parts of cookie-test.pl and show-cookies.pl into a single program which increments the counter cookie value each time we visit the site, displaying its value each time. The first time we visit this CGI program, it sets the cookie's value to 1, the second time we visit, it will set the value to 2 and so on.

You can see a bare-bones attempt at this sort of program in Listing 3. As you can see, the code is fairly straightforward. We create an instance of CGI and use the **cookie** method to extract the value of the counter cookie. We increment that value by 1, create a new counter cookie with the updated value and send that value back as part of the header to the program's response. The body of the response contains a short listing of the names and values of each cookie in the system.

Each time we invoke update-counter.pl, the user's browser recognizes counter as a cookie with the appropriate host or domain name and path, and thus sends counter as a cookie with its request. Update-counter.pl grabs the value of the cookie if it exists and sets it to 0 if it does not. It then increments the value of the cookie and creates a new (outgoing) cookie with the counter name and the updated value. This new cookie is included in the headers which update-counter.pl sends to the user's browser, and the value of the cookie is displayed in the body of the response which contains HTML-formatted text.

This program may not seem very useful, but with a few small variations, it could be useful in a plethora of situations. For example, you could ensure that users

only enter a questionnaire once or keep track of how many times they have requested technical support via the Web rather than by telephone. Another possibility might be a web-based quiz game which presented questions one at a time. You could keep track of a user's score with cookies. Alternatively, you could keep track of which questions the user had already seen, so as not to ask the same question twice. You could even keep track of the user's high score, giving a special message if and when the user achieves a new high.

Using Cookies and Databases

If we were interested in keeping track of multiple values, we could simply create a number of separate cookies. The cookie specification indicates that each host or domain can store up to 20 cookies. Except for a note in the CGI.pm documentation indicating that some versions of Netscape Navigator place a much lower limit on this number, in practice, storing multiple cookies is easily possible.

Just because we can do something does not mean that we should do it, and using multiple cookies is often the wrong approach to a problem. Sites interested in keeping track of several variables for each user have begun to use cookies not to store data about the user, but rather a unique index into a table stored in a relational database. (For more information on relational databases and SQL, see the September, October and November 1997 installments of *At the Forge*.)

How do we do this? First, we create a table in our relational database that gives a unique identifier (known as a primary key) to each row in the table. For example, if we want to keep track of each user's first name and favorite color, we can create a table using the following statements in SQL:

```
create table user_table
   (user_id mediumint auto_increment primary key,
   user_name varchar(60) not null,
   user_color varchar(10) not null);
```

With the above table created in our database, we can create an HTML form into which a user enters his or her name and favorite color (see Listing 4).

Now that we have an HTML form that allows users to submit their name and favorite color to a CGI program, we need to write that program, submit-cookie.pl (see Listing 5). The program first checks to see if the user already has a cookie; if so, it simply updates the existing elements in the user table. A more robust version of this program would check to see if an entry in the table really existed or if the cookie value was not valid for our site.

If no user_id cookie exists, submit-cookie.pl needs to create a new entry in the database table and return a new cookie assignment to the user's browser. We thus insert a new row into the table whose values depend on the values submitted from the HTML form. When we have completed sending our SQL query to the server, we ask the server for the unique ID used when inserting our row, which serves as a user ID and is stored in the user_id cookie by the user's browser. We get this value by using the Mysql **insert_id** method, which tells us exactly this piece of information. Once we have this information, we create a new cookie and return it as part of the HTTP headers in the response to the user.

In either case—whether we create a row in the table or update an existing row—the user is presented with a link to homepage.pl (see Listing 6), a personalized home page program that displays the information we have collected. Remember, none of this information is stored in the cookie on the user's computer. Rather, the information is stored in a table in our relational database with the index stored in the user's cookie file.

Obviously, storing the user's name and favorite color are just examples. A site could allow users to indicate a set of preferences and use a database to choose graphics, text and even hyperlinks based on those preferences.

Conclusion

That's about it for our gentle introduction to cookies and what they can do for you and your web site. There are a few elements of cookie creation and administration that I did not go into, such as expiration dates and security, but those are easily understood after reading one or more of the specifications mentioned in Resources. Suffice it to say that anyone interested in keeping the values of cookies past the current invocation of a browser must handle expiration dates, since cookies created without them only last until the user quits the browser.

As we have seen, cookies provide us with a sense of state that would otherwise be impossible to manage. It isn't hard to think of ways in which to use cookies, whether it be in the creation of personalized home pages, shopping carts for purchasing items on the Web or determining whether a user has previously visited a site. With these types of tools, your sites can become even more interesting for visitors without a lot of extra work.

Resources

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

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #45, January 1998

Readers sound off.

Best of Tech

In the September issue's *Best of Technical Support* column, I was particularly interested in two of the letters.

John Barnitz wrote that he has problems with up and down-spinning disks. If his disk is a SCSI disk, it can, in many cases (e.g., Adaptec-Controller), be switched off in the controller setup. Another method on modern motherboards is to disable some of the power-saving settings.

Are Tysland wrote of problems with **su** and security. I tried **sudo** for a while, but now I am using **su1** (also on sunsite). Using sudo, it is only possible to allow or disallow a command for a user or a specified group of users. With su1 you can control the parameters, e.g., user A can only mount CDs while user B or group B can mount disks. You can control nearly every parameter and command with this tool. I don't believe in a separate group for su users because you have to switch the group before using it. If you don't switch back after using su, you will probably have a problem.

—Martin Fuerstenau, Bilm, Germany tomcat@hannover.sgh-net.de

Grundig TV

The article *Grundig TV-Communications* by Ted Kenney in issue 42 contains a small mistake in the last sentence of the last paragraph on page 54. DPT didn't begin to distribute a Linux driver developed by a third party. What they did, was start listing Linux as a supported OS and linking their web site to my EATA web pages. (I am the author of the EATA-DMA driver for the DPT controllers.)

—Michael Neuffer neuffer@trudi.zdv.Uni-Mainz.DE

Registering in the U.S. Domain

I'd just like to say how much I enjoyed the article by R. K. Owen in the July 1997 *LJ*, *Registering in the U.S. Domain (For Free)*. Using Mr. Owen's carefully spelled out procedures, I tried to register a domain with the berkeley.ca.us "delegated authority", Cliff Frost cliff@ack.berkeley.edu. The response was totally underwhelming, and months later I still do not have my requested domain registered.

This whole US domain registration system seems to be operating at the level of the US Postal Service. (Hmm. Wait, that's uncharitable. The USPS is a whiz in comparison to these characters.)

For example, this morning I went to the page: http://www.isi.edu/us-domain/ and tried to use the web-based registration form, since the Berkeley delegated authority is not responding to me. Clicking on the "Web Based Registration Template" and arriving at the URL: http://www.isi.edu/cgi-bin/usdomreg/template.pl, I got a "Not Found" message.

It appears that the San Jose, CA delegated authority that R. K. Owen used is better organized. I hope your other readers have better luck using the article's procedures than I did.

Keep the good stuff flowing from *Linux Journal*.

—Robert Lynch rmlynch@best.com

E-mail Correction

The October issue is great, but there is a small error—my e-mail address on page 33 is wrong [*Internet Programming with Python* book review]. Instead of djohnson@olympus.net, it should be dwj@aaronsrod.com.

—Dwight Johnson dwj@aaronsrod.com

October Issue

Now you guys have really done it—I mean Issue 42, October 1997.

A smart looking cover, articles packed with answers and numerous relevant reviews, all for five bucks. It's going to take me hours to pore over this thing. *Linux Journal* keeps improving. Keep it up. I may even get around to subscribing one day.

—John Whipple johnboy@sonic.net

Xforms and a.out

I would like to correct the assertion of the responder who claims that Xforms is not available for Linux in a.out format. [*Letters to the Editor*, October 1997, John Brown, "XForms Article"]

The FTP site ftp://einstein.phys.uwm.edu/pub/xforms/ contains a linux/ directory with two subdirectories, one that contains the ELF distribution and one that contains the a.out distribution. Both the old release 0.81 and the newer 0.86 are represented in both formats.

Furthermore, the test subdirectory contains the latest test release (0.87.2) in both a.out and ELF formats, thus refuting the assertion that the developers will not support a.out any more.

—Martin John Bartlett martin@nitram.demon.co.uk

xmotd

I thoroughly enjoyed the article by Luis A. Fernanades entitled, *xmotd: Writing Free Software* [October, 1997]. It was a pleasure to read about a programmer contributing to Free Software and supporting other users. This is the very reason that I switched to Linux. Keep articles like that one coming.

—Steven J. Hill sjhill71@inav.net

Nice Article on Pgfs

The *LJ* article on Pgfs is really great—nicely written and very informative. [*Pgfs: The PostGres File System*, Brian Bartholomew, October 1997.] A very nice example of problem solving with our favorite OS. Thanks for including information on what didn't work; it was just as interesting as what did.

LJ is quickly becoming my favorite magazine.

—Eric C. Newton ecn@smart.net

Alpha Questions

Table 1 on page 30 of the October issue is hardly a "Summary of Alpha Chip Family". [*Linux and the Alpha*, David Mosberger]

David Mosberger surprised me when he stated in the Alpha article that gettimeofday() "returns the current real time at a resolution of typically one timer tick (about one millisecond on the Alpha)". The gettimeofday() API allows

a resolution of one microsecond, and Linux on the i386 gives you that (courtesy of a hardware counter that is incremented at approximately one megahertz). A quick look at the 2.0.29 kernel sources (arch/*/kernel/time.c) suggests to me that gettimeofday() gives you high resolution on the i386, m68k, mips and ppc, but not on the Alpha or SPARC. Are there plans to fix this inequity?

—James R. Van Zandt jrv@vanzandt.mv.com

No, it's not. Correct table is printed below.

I think you raise three different questions:<\n>1. What does _typical_ resolution mean?2. Would gettimeofday() with one microsecond be sufficient for the kinds ofmeasurements I wanted to discuss?3. Will Linux/Alpha support resolutions finer than 1 clock tick?

The answers:

1.To user X, “typical” is what user X happens to own, I suppose. Seriously though, the traditional gettimeofday() implementation provided timer-tick resolution. The API obviously always has allowed for up to microsecond resolution, but that's not relevant for actual implementations. If you're writing a portable program, you cannot rely on gettimeofday() returning better than timer-tick resolution, hence my claim of this being “typical”.

2. One microsecond resolution is far from sufficient for modern CPUs. Even on a (relatively slow) 200MHz P6 this would correspond to 200 cycles, so it would not be possible to measure low-level events such as primary cache-misses. Even if the resolution were sufficient, keep in mind that getimeofday() typically involves a system call. For example, on a 200MHz P6, it appears to take on the order of four microseconds just to execute a pair of gettimeofday() calls. In summary, there are plenty of scenarios where a CPU cycle counter is advantageous over gettimeofday(). Of course, the converse is also true. What I'm saying is that both techniques have their place for the time being.

3. I don't know. 2.1.xx may already do that, but I haven't had a chance to keep track of recent kernel development.

—David Mosberger davidm@azstarnet.com

I2O and *Stop the Presses*

I have just read Phil Hughes' *Stop the Presses* in the October issue of *LJ*. Being a manufacturer of intelligent multiport cards, Cyclades has been following I2O for a while.

I2O has been around for a few years now. It started to gain momentum only this year, due to the persistence of Intel. SCO, Microsoft and Novell seem to be going to support it. At this point, there are no real implementations of I2O, and the standards are only now becoming stable. I have doubts whether this will actually become real or not. The idea of a standard architecture is good, but I2O is a very heavy interface designed to sell Intel processors.

Besides the fact that you have to pay to get the specs (not necessarily bad, PCI does the same thing), there is another catch. The I2O specs were designed so that there is only one possible CPU to be used on a I2O card: the Intel 960RP. More, only Wind River (working with Intel) has a software implementation of I2O messaging. Intel bundles a runtime license of IxWorks (the RTOS that runs the I2O messaging inside the board) with every 960RP.

Thus, I2O is not an “open” standard. It is an attempt by Intel to standardize intelligent I/O around the Intel 960RP and Wind River's IxWorks.

With I2O gaining some momentum, other hardware companies (PLX is one of them) are designing PCI chips compliant with I2O messaging. But, still, the choices are limited, and the viability of I2O is still to be verified.

Phil also mentioned the “Open Hardware” initiative. Cyclades was the first company to certify products for Open Hardware and is, still, the only one.

—Marcio Saito, Cyclades Corporation marcio@cyclades.com

Archive Index Issue Table of Contents

Advanced search

The Beowulf Project

Marjorie Richardson

Issue #45, January 1998

With the Beowulf project, NASA has provided the Linux community with the opportunity to spread into scientific areas needing big computing power.

Beowulf is one of the most exciting projects using Linux today. Originating from the Center of Excellence and Information Sciences (CESDIS) at the NASA-Goddard Space Center in Maryland, the project's mission statement is:

> Beowulf is a project to produce the software for off-the-shelf clustered workstations based on commodity PC-class hardware, a high-bandwidth internal network and the Linux operating system.

The Beowulf project was conceived by Dr. Thomas Sterling, Chief Scientist, CESDIS. Donald Becker, a Scientist at CESDIS, wrote the fast-Ethernet drivers needed for the Beowulf-class clusters and incorporated them into Linux. One of NASA's imperatives has always been to share technology with universities and industries. With the Beowulf project, NASA has provided the Linux community with the opportunity to spread into scientific areas needing *big* computing power.

Our cover is a picture of the Beowulf-class cluster, Loki, used at Los Alamos National Laboratory. For more information about this cluster and the Hyglac cluster at Caltech, see our feature article *I'm Not Going to Pay a Lot for This Supercomputer!* by Jim Hill, Michael Warren and Patrick Goda. There are also clusters of this type at Drexel and Clemson Universities.

In February

Next month, our focus will be on databases, and we have articles on development, PostgressSQL, SQL in Python and a free database called Qddb. I think you will find them all enlightening. We will also feature an article about Digital Domain, a production studio that does digital special effects for the

movies. Digital Domain used 105 Alphas running Linux and connected by fast Ethernet to create many of the effects used in the upcoming motion picture *Titanic*.

7th USENIX Security Symposium

The goal of this symposium is to bring together researchers, practitioners, system programmers and others interested in the latest advances in security and applications of cryptography. This will be a four-day symposium with two days of tutorials, followed by two days of refereed paper presentations, invited talks, works-in-progress presentations and panel discussions. It will be held January 26-29, 1998 at the Marriott Hotel in San Antonio, Texas. For more information contact the USENIX Conference Office via e-mail at conference@usenix.org.

Linux Resources on the Web

We've updated our Linux Resources web pages—check them out at http://www.linuxresources.com/. In addition to the usual sections pointing to Linux distributions, FAQs, HOWTOs and Newsgroups, we have:

- **Linux Speaker's Bureau**: a list of people willing to give Linux talks and their areas of expertise.
- **Employment**: a place to go to find a job or the right person for a position utilizing Linux.
- **Business Connection**: information on how to promote Linux in the workplace.
- **Linux Projects**: a list of current projects being developed in Linux and a message board for posting ideas for projects.
- **G.L.U.E.**: Groups of Linux Users Everywhere, provides information on starting a LUG, setting up trade shows and more.
- **Linux Library and News**: places to find the latest information about what's happening in Linux.

We also are providing space for people in the Linux community to post their information about Linux, so that it can be found in one area rather than scattered over the web. Write to info@linuxjournal.com if you wish to make use of this space. Because we wish this Linux Resources area to be a community effort to promote Linux, we have kept it non-vendor specific and advertisement free.

Subscription Fulfillment

Several months ago a light bulb went off and we had what we thought was a great idea—turn subscriptions over to a fulfillment house. Our Associate Publisher did a lot of research to find companies that performed this service and that could accept e-mail orders. We then picked one that seemed to best fit our needs. In September, we turned subscriptions over to Superior Fulfillment in Duluth, Minnesota, expecting to no longer have to worry about this aspect of the magazine. Well, we were wrong. Murphy's Law took over and everything that could go wrong did. Superior kept delaying the date when they would have all of our orders entered into their system. It was time to get label information to the printer, and we didn't have it. We began to get angry letters from new subscribers. Finally, we had to give up on being able to fix the situation long distance. Superior returned all of the subscription orders to us, and the majority of *LJ*'s staff spent the next couple of weeks entering orders and writing letters to our subscribers. The insert cards in the November and December issues of *LJ* all have the Duluth, MN address, but now it's back to Seattle.

We are most sorry for the inconvenience that this situation has caused our customers and hope that you will all forgive us. To us, our subscribers are the most important people in the world, and we wish to give you the best possible service. Thanks to all of you who showed us such patience and understanding during the time period when it seemed like all orders were lost.

Archive Index Issue Table of Contents

Advanced search

LISA'97 Conference

Phil Hughes

Issue #45, January 1998

The conference was very Linux-aware.

I am writing this column from the LISA'97 conference in San Diego. LISA is a join effort of USENIX and SAGE (Systems Administrators Guild). USENIX conferences tend to be attended by technical computer professionals, dressed in t-shirts, gathering to learn from talks and by talking to their peers, to see what the vendors have to offer and to have a good time. LISA was no exception.

There were close to 100 booths with many Linux and Linux-friendly vendors including Caldera, Digital Equipment Corporation, Enhanced Software Technologies, Red Hat Software and Walnut Creek CD-ROM. I also had the pleasure of handing the people in the Ross Technologies booth a copy of the October *Linux Journal* Issue 42, the one with the Ross HyperSPARC on the cover.

The conference was very Linux-aware. Vendors who did not have a Linux sign in their booth responded to a gentle prod from me with their own suggestion of adding one.

COAS

I arrived on Tuesday evening in time to attend a BoF (Birds of a Feather) session put on by Caldera. At the BoF, Caldera announced COAS, the Caldera Open Administration System. Tim Bird of Caldera and Olaf Kirch of LST in Germany—soon to be Caldera GmbH—discussed the technical details of the project and offered a demonstration.

The quick summary is that COAS is a unified approach to systems administration. While other Linux vendors such as Yggdrasil and Red Hat have offered some tools for systems administration, none have offered a unified approach. The system needs to handle administration of everything from users

to firewalls and have the necessary interfaces so system administrators will be happy with the choices. Also, for now the system needs to interoperate with vi, “the universal administration tool”, as Tim Bird called it.

Caldera decided to develop COAS because its customers demanded it. To quote Tim again, “[This tool] is needed to make Linux play in the same space as the big boys”. Another important part of Caldera's plans are that COAS will fall under the GPL; that is, anyone will be able to contribute to the effort.

Will COAS work? The only non-Caldera person who has looked at the code so far is Bruce Perens, head of the Debian project. I talked to Bruce, and he is pleased with what he has seen and feels the framework is there to do multiple system administration, a concern of many including myself.

I feel COAS is a solution that is needed, and it sounds like Caldera wants to work with the whole Linux community to turn it into the right system. I have added a discussion channel on *Linux Journal* web site (http://www.linuxjournal.com/discussions.html) to help encourage everyone to get involved. Feel free to chip in and express yourself on the ideas being presented.

The Rest of the Show

For those of you who think Linus is always out drinking beer, I can confirm that this is not the case. Linus, Dan Quinlan and I shared a bottle of wine—just another way to show the versatility of the Linux community.

Speaking of Dan Quinlan, he is the author of the Linux File System Standard and has just released a new version. (See http://www.linuxresources.com/.) There are some important changes, so you should probably take a look at it. He will also be writing a follow-up article on the standard for *LJ*.

The terminal room was a very pretty site with people typing away on 30 Debian Linux systems. It would have been more fun if this had been a Win95 conference, but it did prove that Linux can rush in and solve a problem.

All in all, this was another great USENIX show. It made me want to take off my publisher hat, put on my nerd hat and invest some time in attending the tutorials and sessions. There is a lot to learn and USENIX shows are an excellent place to do that learning.

Finally, the best rumor of the show: Linux is now running on SGI hardware. Expect to hear more about this port in the next few months. There are some code licensing issues that need to be addressed before the port can be released for public use, but it sounds like it will happen.

LINUX JOURNAL

Need More Info?

Bill W. Cunningham

Issue #45, January 1998

Here's how to get the information you need using GNU's hypertext system called info.

Have you ever been reading along in a man page and come across something like, "**SEE ALSO gdb entry in info**" and wondered what it meant? Still perplexed, did you pick up *Running Linux* or *Unix Unleashed* and fail to find one word about **info**? If so, this article is for you.

Background

info (information) is the GNU project's hypertext system filling the gap between the man pages and printed reference books. **info** is a viewer for info files. Without going into excruciating detail, info files are rigidly formatted text files that are processed with the **texinfo** program, and then compiled by **makeinfo**. They end up in gzip format and usually live in /usr/info/ (at least on Slackware 96). The info program's job is to uncompress them and present them to the reader. Info files can also be viewed with Emacs, but this article will focus only on the stand-alone info program.

The info files make up a cross-referenced, virtual hierarchy of documentation. At the top of this hierarchy is the directory node. If you run info without arguments at your command prompt, the directory node opens on your console. The directory node serves as the master index for all info documentation on your computer and provides a point of entry into all these other files. The files can also be accessed directly, without going to the directory node first. For example, I can type **info make** at my prompt and go directly to the documentation for **make**.

info's documentation is divided into nodes—structures that are basically analogous to paragraphs. A node can be one short sentence or several screens. Nodes are physically stored in files. The info program allows both cross and

vertical links between nodes. Once inside info, you can move from node to node in a random search for data, or you can quickly find and go to specific information. The documentation's level of detail increases as you move down the hierarchy tree. At any given level of detail, you can follow cross references to related topics at approximately the same level of detail.

Of course, someone has to write the documentation for it to be available, and not all Linux software has data available for info. Most info data deals with gcc, g++, gdb, make and other topics related to programming and systems administration. In other words, lots of info documentation exists for older software; this makes sense as info has been around for a while. Users researching these areas are fortunate in that they will find a gold mine of information that's as easy to read as a man page.

Hands On

Essential GNU Info Reference Card

At this point, you may wish to look at the Reference Card and, with these essential commands in mind, try navigating info on your Linux system. Just enter **info** at the prompt and start reading. I recommend entering **h** at the directory node and working through the on-line tutorial first thing. The whole course only takes about half an hour. The only confusing point I found was that info files can only be printed from within Emacs (the M-x print-node option). Neither the tutorial nor the info man page makes this clear, but I found it to be true.

New Info Files

What happens when you get a new software package, **mgetty** for instance, that includes several info files as part of its native documentation? You could just leave those files where they are and accept the fact that you would have to run **info mgetty** to view them. Or you could install the new files into the existing hierarchy. You don't even have to move them to the default info directory.

First, note the location of the new info files. I'll stick with mgetty as an example. These files were installed in /usr/local/info/. In this case, the master info file is named /usr/local/info/mgetty.info. Master info files always end with the .info suffix.

Next, add mgetty's entry to info's directory file. To do this, edit the file /usr/info/dir as root. Move down to an appropriate location for the new entry. I chose “Miscellaneous commands and documents”, the next to last section. Menu entries in the dir file consist of three sections. The first is the menu entry name, starting with an asterisk and followed by a colon. The second is the menu file

name in parentheses, followed by a period. (The .info suffix isn't needed here, even though it's part of the actual file name.) The third section is the package description, which contains the following entry for mgetty:

```
*mgetty:        (mgetty).       mgetty documentation.
```

Add this line to the /usr/info/dir file and exit saving the file. Finally, info has to be able to find mgetty's info files, since they're not in the default directory, /usr/info/. Although we could specify (/usr/local/info/mgetty) in our entry's second part, that would really make the dir file ugly. The most aesthetically pleasing solution is to set the INFOPATH variable. The directories in INFOPATH are searched by the info program for data files. So, add the line:

```
export INFOPATH=/usr/info:/usr/local/info
```

to /etc/profile, and on the next login, all users will be able to access mgetty's info documentation just as if it had always been part of the hierarchy. If mgetty is ever upgraded—no problem—info finds the new documentation with no further adjustments.

Suggestions

I wish that movement within nodes responded to the page up and page down keys. That would be more intuitive for most people. Color syntax highlighting might also be a nice touch. Basically, info works fine just as it is.

Conclusions

If you're used to hypertext in the sense of HTML and Netscape, info may seem spartan by comparison. Indeed, a lot of documentation these days is in HTML form, and info may quietly fade away in Netscape's wake. But keep in mind that info's man page was first copyrighted in 1989—quite a while before hypertext in the present sense came about. For an all-ASCII system that's going on 10 years old, it does a pretty darn good job.

Gunnery Sergeant Bill Cunningham, US Marine Corps, got his BS in Computer Science in 1991. He is the LAN Chief for Marine Forces South, Panama Canal Zone. In his leisure time, he plays with his 3 fine kids and helps his wonderful (and expecting) wife pick up coconuts that fall into their yard. He can be reached via e-mail at bill@lancomm.mfs.usmc.mil.

Kill: The Command to End All Commands

Dean Provins

Issue #45, January 1998

Need to get rid of a job that's gotten into a loop and refuses to end? Here's a command that will take care of the problem.

Linux is a powerful operating system. With its demand-paged memory management and swap file facility, it lets you start as many processes as you choose. Of course, that number is subject to overall system memory capacity (physical memory plus swap) and your CPU's ability to perform all the tasks you have requested. Starting processes is easy, and when things slow to a crawl, stopping them is just as easy.

The Linux **kill** command is one of two that will meet your need when you grow tired of waiting for a process to terminate. With it, you can, in the words of my 1992 Linux Programmer Manual, terminate a process with extreme prejudice. All you need to know is a number called the process PID. Note that kill doesn't always terminate another process. In essence, kill sends a signal to a specified process. If that signal is not caught and handled by the process (not all can be), the process is terminated. All of the resources that were in use by the process are released for use by other running processes.

Processes and PIDs

What are processes, PIDs and signals? How are they discovered?

Recall that Linux is a multi-tasking operating system. When Linux boots, it starts a program called **init**, which in turn starts other programs. Many of these are background tasks like **update**, which periodically flushes data to the disk. Another example is **getty**, which watches a serial port for some sign of activity. A more visible example is the shell you use to perform useful work. It runs in the foreground, which means that it waits on your keystrokes. Each copy of each program running on your system is called a *process*.

Just as the US government passes out Social Security Numbers (we use Social Insurance Numbers here in Canada) to uniquely identify each individual, Linux assigns each process a unique number as an identifier. This number is called the *process ID* or *PID*.

When a process is started, it is given the next available PID, and when it terminates, its PID is released for eventual re-use. To determine the PID of any process belonging to you, enter **ps** at the prompt. The **ps** command will print, for each of your processes, a line containing the process's PID, the amount of time the process has used and the command with which the process was started. The output from **ps** looks like:

```
  PID  TT STAT   TIME COMMAND
  6651  p0 S      0:01 -ksh<\n>
  6661  p1 S      0:00 -ksh
  6738  p2 S      0:00 -ksh
  6746  p2 S      0:00 wheel
  6747  p2 S      0:00 wheel
  7002  p0 S      0:01 elm
  7193  p1 R      0:00 ps
```

Signals

Signals are a form of process communication. Because they can come from another process, the kernel or the process itself, they might be better thought of as events that occur as a program runs. A crude example might be the bell most of us remember from our early days in school; when the bell rang, we reacted by switching from playful children to industrious students.

The signals we will use below are the termination signal **SIGTERM**, the interrupt signal **SIGINT** and the kill signal **SIGKILL**. These signals usually occur because another process sent them. You probably already use one of them; typing ctrl-c sends the interrupt signal **SIGINT** to your current foreground process. Other signals—such as **SIGPIPE**, which is sent to a process writing to a broken pipe—usually come from the kernel. There are about 30 signals, all of which can be referred to by numbers or by names, but the numbers change between platforms and some signals are unavailable on some platforms. The complete list of signals can be found on the signal(7) manual page; enter **man 7 signal** to see it or enter **kill -1** for a short version of this list.

For each signal there is a default action, almost all of which terminate the process. For most signals, a program may specify another action—this is called *catching* or *handling* the signal—or may specify that no action occurs, which is called *ignoring* the signal. The signal **SIGKILL** cannot be caught or ignored; it always terminates processes.

For example, suppose you use **cat** to list a large text file without first determining the size of the file. Instead of watching hundreds, perhaps

thousands of lines scroll by too quickly to read, you send the **cat** process the interrupt signal by pressing Ctrl-c. Fortunately, **cat** was not programmed to catch **SIGINT**, and the **cat** process is terminated immediately.

Using kill

Suppose we had inadvertently started **cat** in the background; Ctrl-c would be ineffective, because the signal wouldn't get to the **cat** process. So we need to send it a signal some other way. With the command **kill**, you can send any signal to any process you own. The command's syntax is:

```
kill -SIGNAL
```

If no signal is specified on the **kill** command line, the terminate signal **SIGTERM** (default) is sent. This will normally terminate the process in question. If it fails to do so—that is, **SIGTERM** was caught or ignored—you can send the signal **SIGKILL**, which will always terminate the process.

Thus, we might do the following to terminate our runaway **cat** process commands. First determine the PID:

```
$ ps
  PID  TT STAT   TIME COMMAND
 2037  p0 S      0:01 cat
```

Now, kill process 2037, which is the **cat** process:

```
kill 20371
```

If **cat** had been written to catch **SIGTERM**, we would have to use a signal that cannot be caught or ignored.

```
kill -SIGKILL 20371
```

In addition to killing errant processes, kill can be used to inform processes that the status of something has changed. For example, suppose you are writing a program and you wish to have it change its mode of operation on the occurrence of some external event. By coding what is called an “interrupt handler” in your program, you can have it catch any number of signals which have meaning to you. In particular, you might choose SIGUSR1 or SIGUSR2, which are non-specific. By sending your chosen signal, you can make your program aware of the change in circumstances, so it can proceed into its alternate mode of operation.

When you use kill, the desired signal is sent only to processes you own (that is, processes that you invoked). This prevents inadvertent termination of the wrong process. The exception is that the superuser (root) can use kill to send a

signal to any process. Similarly, any process owned by root can send a signal to any other process.

An orderly shutdown of your system can occur in this way. While the kill command is not used at shutdown, the equivalent system call **kill(2)** is used to terminate everything. This guarantees that no files are left open and that all buffers are written to disk. For a description of **kill(2)**, enter **man 2 kill** at the prompt.

A related command is **killall**, which takes the name of the process as an argument rather than the process ID (PID). (Some versions of kill can take process names too.) This is a convenient way to terminate all processes with the same name. If a path is used to identify the process to be signaled, only the processes executing that particular file are selected. In addition, you can ask to be consulted before **killall** kills a particular process, and you can receive confirmation that the signal was actually sent.

Although full details are listed in the **man** page, an example may be useful here. Suppose you have two programs that are different but have the same name—perhaps different release levels. In order to be different and have the same name, they must be stored in different directories. Assume they have the name **sample_prog**, but one is stored in /usr/a and the other in /usr/b. Entering **ps** gives the following output:

```
  PID TTY STAT  TIME COMMAND
123 pp0   S     0:03  /usr/a/sample_prog
124 pp1   R     0:02  /usr/b/sample_prog
```

The following commands perform different actions:

```
# To kill both processes
killall sample_prog
# To kill only process 123
killall /usr/a/sample_prog
```

Conclusion

In summary, the **kill** and **killall** commands can be useful tools to control the execution of processes on your Linux system. In combination with other tools described in previous “Take Command” columns, they will allow you to become true masters of a very powerful desktop appliance. For specific information on their very few options, and for a description of the signals they can invoke, read the relevant manual pages (enter **man kill** or **man killall** at the prompt).

Dean Provins is a professional geophysicist and licensed Amateur Radio operator (VE6CTA) in Calgary, Alberta. He has used Unix systems since the mid 1980's and Linux since January, 1993, when he read an article about it in the Calgary Unix Users' Group newsletter. Dean uses Linux as a development system for geophysical software, and as a text processing system for newsletter and other articles. He is currently developing a Linux application to view scanned images of articles published in the American Radio Relay League's monthly journal *QST*. He can be reached at provinsd@cuug.ab.ca.

Archive Index Issue Table of Contents

Advanced search

Linux at Rancho Santiago College

Steve Moritsugu

Issue #45, January 1998

Linux is being used to teach Computer Science classes at a community college in Santa Ana, California.

Roughly twice a year, I find myself combing the Unix job ads in the *LA Times*. Luckily, this does not mean I am frequently unemployed. In fact, in addition to my full-time job integrating, supporting and teaching Unix, I am also a part-time instructor at Rancho Santiago, a community college in Santa Ana, California. For the first hour of the first class of every semester, the parking lots are a jungle, and students are struggling to find the right buildings and classrooms. I use this hour to discuss the recent Unix job ads and "buzz words". I cover what students will learn in my class and what additional skills employers list most often in these ads. In January 97, employers most often wanted Unix plus C programming or Unix plus Windows NT (networking). Rarely do ads mention a specific Unix operating system type such as HP, Sun or Linux.

Unix Like a Car Radio?

I use Linux extensively in my classes. I also use the free version of SCO Unix. I teach a series of three full-semester Unix classes available at Rancho:

1. CMPR 141: Unix Operating System
2. CMPR 241: Unix System Administration
3. CMPR 242: Advanced Unix Shell Scripts

In each of these classes, I give my car radio analogy. There are many different styles of car radios, each with a slightly different placement of knobs and controls; yet when most people get into a strange car, they can turn on the radio, adjust the volume, change the station and maybe set their favorite stations. In the same way, Unix on different computer platforms has variations in the basic commands, e.g., how you log off or how you display a file one page at a time. There is no need to become disoriented when you face a strange car radio, oops ... I mean when you face a new variant of Unix, especially if you are on the Internet, using TELNET to access one host or another. While Linux has peculiarities that you don't find on other flavors of Unix, it is not alone. Every Unix has some peculiarities. For me, the important issue is that Linux supports all the standard Unix commands and shell features.

Why Linux?

When I first started teaching at Rancho, my biggest problem was the fact that students had no access to Unix outside of my three-hour once-per-week class. The school has a computer lab, but none of the systems there had access to Unix. I considered several alternatives before deciding on Linux. There are packages that allow you to run Unix utilities under MS-DOS. I decided against using these, because the problems that arose were not really Unix issues, such as MS-DOS not supporting upper and lower case file names or Unix permissions. There was another flavor of Unix that could be loaded from floppy, but that company is no longer in business. The free version of SCO was not available at the time and, since it is only a 2-user version, it must be installed at each station which is difficult in a lab situation.

I also considered freeBSD. However, I eventually chose to use Linux in my classroom for the following reasons:

- Linux acts like a standard version of Unix and the problems that arise are true Unix issues (not MS-DOS issues).
- Linux has a large body of documentation and HOWTOs that allow me to customize the package as needed.
- Linux licensing allows me to make the software freely available to students.
- Linux evolves continuously, supporting new hardware as it becomes available.
- Internet ISPs use Linux as e-mail and/or domain name servers.
- Linux is strongly represented in the technical book stores under both the Unix and Internet categories.
- There are a number of different Linux distributions, fostering healthy competition and adding value.

All of the above show that Linux is a thriving variant of Unix, worth studying in its own right, as well as a useful teaching tool.

On the other hand, I also warn my students that Linux is not yet fully accepted as a commercial grade of Unix by all employers. I also use the free SCO Unix in my classroom so that students can see variations in Unix side-by-side, and so they can list SCO Unix as well as Linux in their resumes.

Floppy-Based Linux

For my students, I have created a distribution of Linux which I call Floppy-Based Linux. It runs completely from floppy and never accesses the hard disk, yet it supports all the standard Unix commands and man pages. It can also use TELNET and FTP to access the Linux server running in the classroom and can print to network printers in the classroom and the computer lab. Of course, this is slower than running from hard disk, but it provides a number of advantages in my environment. My students are often computer neophytes. I need a bullet-proof environment where they can freely make mistakes. Thus, I ruled out any distributions that would require them to load software onto the hard disk, even the UMSDOS file system. I want my students to be able to work on Unix at home, at work or in the school computer lab since I believe that accessing Unix hands-on is essential to learning Unix. At the same time, I could not take a chance that a mistake on their part might alter or corrupt their home computer, work computer or lab computer.

My students can run Floppy-Based Linux on any PC with 8MB of RAM and a 3.5-inch floppy drive. The floppy diskettes are write-protected when in use, so they cannot be corrupted. The Linux kernel has been rebuilt, so that it has no drivers for IDE or SCSI hard disks to ensure that a student cannot alter the hard disk. At first, I started with the Slackware 3.0 boot diskettes and RAM disk, but I found that these are not in ELF format so I could not add some of the commands I wanted. Following the boot disk HOWTO, I created my own boot diskette and 3MB RAM disk. After loading the RAM disk, an rc script prompts the student to

insert a “Supplement” diskette, which loads more files into the root file system in memory. Students then mount a “Utils” diskette which gives them access to many more Unix commands and man pages.

Students can use Floppy-Based Linux for homework assignments outside of the classroom. Inside the classroom, there are 18 Windows PCs, one Linux Server and one free SCO Unix system. Students bring up Floppy-Based Linux on the Windows PCs and then download new homework assignments from the Linux server using FTP. They also upload their completed homework to the Linux server. They can print to a network printer in the classroom. I have also used variants of Floppy-Based Linux to debug network problems and to make image backups of a Windows hard-disk partition using **dd** (device-to-device copy) “piped” to **rsh** (remote shell) to save the data on another system in the network or its tape drive. (Hard disk access was enabled in order to do this.)

Electronic Blackboard

I have taught Unix and networking seminars and training sessions nationwide. There is one teaching technique I use in these sessions and also in my classroom that I am asked about most frequently. I call it my “electronic blackboard”. It allows me to broadcast everything that appears on my station to all Unix stations in the room. Students can watch me enter commands, backspace, make corrections and use command-line editing to develop long, complicated pipe sequences in gradual steps. Instead of writing on a blackboard, I use vi to put any notes into a file. Students see the notes as I type them, as well as how I use vi, and at the end of class I can print the file as a handout. (It helps that I am a speedy typist.)

This broadcasting technique is scalable in that I can broadcast to one station as easily as to 200 or 500 stations. I can do this broadcasting using two simple Unix commands that are available on all flavors of Unix including Linux. No special software is required. Can you figure out how I do it? Here is the answer. At my station, I first run this command:

```
telnet localhost | tee /ttt
```

This gives me a new Unix login. All output and the echo of everything I type will now be saved in the file **/ttt**. (If you don't have networking set up, you can do the same thing using **cu**, or call up, and a loop-back cable between two serial ports.)

At each of the student stations, run this command:

```
tail -f /ttt
```

tail will run until aborted, reading every new keystroke added to file **/ttt** and displaying it on each student's screen. Now each screen perfectly reproduces what appears on my screen. I can also automate this by writing a script to start all the **tail<\!s>-f** commands at one time. Note: each student station must be set up to use the same TERM variable as I am using at my teaching station, since escape sequences and graphics will also be broadcast.

Advanced Regular Expression Lesson

In closing, I would like to leave you with a lesson that I teach in advanced regular expressions, which applies to Linux as well as other flavors of Unix. The rule is: "to simplify line-oriented regular expressions, drop unbounded *or more* wild cards." This is analogous to simplifying unnecessarily complex fractions in mathematics. The simplified expression is more efficient to implement and easier to understand. I find that most students understand advanced regular expressions much better once they understand this rule.

In regular expressions there are two wild cards that I call "or more" wild cards, as shown in these two examples:

```
grep  'aa*bc'  file
```

Search for and display any lines in **file** that contain one or more instances of a's, then b, then c.

```
grep -E 'abc\{10,\}' file
```

Search for and display any lines in file which contain a, then b, then 10 or more instances of c.

Both of these "or more" wild cards are unbounded, because the "or more" wild card occurs either at the start or the end of the regular expression pattern.

Contains "or more"	Better Pattern
1. aa*bc	abc
2. abc\{10,\}	abc\{10\}
3. zaa*bc	zaa*bc (same)
4. abc\{10,\}$	abc\{10,\}$ (same)

In example #1 above, the “or more” wild card is unbounded, so leave it out. Searching for **abc** is more efficient and easier to understand than searching for **aa*bc** and results in the same lines being selected.

In example #2 above, the “or more” wild card is again unbounded, so leave out the comma. To prove that two regular expression patterns are equivalent, we need to show:

1. They do select the same lines. In this case, any line that contains a then b then 10 or more c's also contains a then b then 10 c's.
2. They avoid the same lines. In this case, any line that does not contain a then b then 10 or more c's also does not contain a then b then 10 c's.

In example #3 above, the “or more” is bounded on both sides, so you cannot ignore it. Looking for z, then one or more a's then b then c is different than looking for z then a then b then c.

In example #4 above, the “or more” is bounded on both sides, so you cannot ignore it. The **$** sign indicates the 10 or more c's must occur at the end of the line, which is different than looking for exactly 10 c's at the end of the line.

The **sed** command is not line-oriented, hence this rule does not apply.

```
sed s/aa*bc//g file
```

This command instructs **sed** to delete any segment in **file** that consists of one or more a's, then b, then c.

Command to delete any segment which consists of a then b then c:

```
sed s/abc//g file
```

Since sed is not line oriented, do not drop unbounded “or more” wild cards.

Steve “Mor” Moritsugu (mori@dtrbus.com) is a senior software engineer at DTR Business Systems in Walnut, California. He is also the Unix instructor at Rancho Santiago College in Santa Ana, California.

Archive Index Issue Table of Contents

Advanced search

New Products

Amy Kukuk

Issue #45, January 1998

MaXimum cde Executive Edition v1.0, c-tree Plus v6.7A, MATCOM V3 and more.

MaX*i*mum cde Executive Edition v1.0

X*i* Graphics has released a run-time version of the Common Desktop Environment (CDE) from The Open Group. This new product is called MaX*i*mum cde Executive Edition v1.0, and it incorporates the Accelerated-X Display Server v3.1 and an enhanced CDE v1.0., specifically tuned for Linux. The package features more than 500 graphics adapters. There is also a version available for many of the more popular laptops. The MaX*i*mum cde Executive Edition v1.0 is shipping for $195US.

Contact: X*i* Graphics, 1801 Broadway, Suite 1710, Denver, CO 80202, Phone: 303-298-7478, Fax: 303-298-1406, E-mail: info@xig.com, URL: http://www.xig.com/.

c-tree Plus v6.7A

FairCom Corporation announced the latest release of c-tree Plus v6.7A file handler. The new version offers conditional index support, ISAM enhancements, batch performance, system configuration function, transaction processing enhancements and file path control. c-tree Plus v6.7A is priced at $895US.

Contact: FairCom Corporation, 4006 W. Broadway, Columbia, MO 65203,Phone: 573-445-6833, Fax: 573-445-9698, E-mail: info@faircom.com,URL: http://www.faircom.com/.

MATCOM V3

MathTools Ltd. announces MATCOM V3, a Matlab 5 to C++ compiler. MATCOM V3 compiles Matlab 5 source files (M-Files) to C++ source code. MATCOM can be used to create stand-alone C++ applications with royalty-free distribution. A time limited evaluation version of MATCOM V3 can be downloaded at no cost from the MathTools web site.

Contact: MathTools Ltd., P.O. Box 855, Horsham, PA 19044, Phone: 888-628-4476,Fax: 212-208-4477, E-mail: info@mathtools.com, URL: http://www.mathtools.com/.

Embedded Eiffel

Interactive Software Engineering, Inc. (ISE) is now shipping Embedded Eiffel, an object-oriented application development tool which combines multi-threading and automatic memory management for embedded and real-time applications. This version of ISE's Eiffel 4 software is designed to handle performance and safety requirements. New features include more speed, a small RAM footprint and new object-oriented code.

Contact: Interactive Software Engineering, 270 Storke Road, Second Floor, Goleta, CA 93117, Phone: 805-685-1006, Fax: 805-685-6869, E-mail: info@eiffel.com, URL: http://www.eiffel.com/.

ION (IDL on the Net)

Research Systems, Inc. announced ION (IDL on the Net). ION is a web-server-based system. It is designed for organizations with a need to access, visualize and analyze shared data. ION consists of three components; ION Java Applets, ION Graphic Component Java Classes and ION Java Classes. Price varies depending on the number of concurrent connections to the ION server. It is sold in two, five, ten, fifteen and unlimited-connection license configurations, with prices ranging from $6,995US to $18,995US.

Contact: Research Systems, Inc., 2995 Wilderness Place, Boulder, CO 80301,Phone: 303-786-9900, Fax: 303-786-9909, E-mail: info@rsinc.com,URL: http://www.rsinc.com/.

Audiotrix 3D-XG

Mediatrix Peripherals has announced the release of the Audiotrix 3D-XG sound card for the PC platform. The Audiotrix 3D-XG features 676 voices and 21 drum kits while supplying 32-note polyphony and 16-part multi-timbrality. On-board effects include reverb, chorus and other variations. The product includes an 18-bit DAC for MIDI instrument rendering. The Audiotrix 3D-XG is priced at $295US.

Contact: MediaTrix Peripherals, Inc., 4229 Barlock Street, Sherbrooke, Quebec, CanadaJ1L 2C8, Phone: 819-829-TRIX, Fax: 819-829-5100, E-mail: info@mediatrix.com,URL: http://www.mediatrix.com/.

CCVS 2.0

HKS has announced the release of CCVS 2.0 (Credit Card Verification System). The new version features credit card clearing on workstations using either Mastercard and Global Payment Systems (MAPP) protocol, First Data Resources ETC 7 or Omaha protocol. Other features include support for Address Verification Service (AVS) and leased lines, a Developer's Toolkit for integration with POS, web server and database applications.

Contact: H.K.S., Inc, 4618 Henry Street, Pittsburgh, PA 15213, Phone: 888-457-7836, E-mail: sales@hks.net, URL: http://www.hks.net/.

Archive Index Issue Table of Contents

Advanced search

Securing Networked Applications with SESAME

Paul Ashley

Bradley Broom

Issue #45, January 1998

This article describes the SESAME Security Architecture and how it can be used to secure your networked applications.

Security is becoming increasingly important for Intranet and Internet applications. We hear almost daily of new security incidents on the Internet, sites being penetrated or sensitive data being captured by attackers. There are a number of solutions that together can secure your network.

The first is a firewall with the aim being to control access between networks. Typically they are used at a site's Internet connection to control who can get in and out between the site's internal network and the Internet. In some intranets, firewalls are also used to control access within the network.

However, firewalls are not the end of the solution. Firewalls do not protect applications behind the firewall from internal attack or secure applications that need to communicate through the firewall. We require a method to secure applications; the client and server portions of an application need to verify each other's identity (authentication), and data being transferred across networks must be protected from unauthorised viewing (confidentiality protected) or modification (integrity protected).

We have recently ported SESAME to Linux, and it provides a wonderful tool for anyone who needs to secure their applications on Linux. SESAME is also available on a range of other Unix platforms, so your Linux systems will be able to work with these, too.

Kerberos

The SESAME Security Architecture is an extension of Kerberos, arguably the most famous and most successful architecture to date for securing networks (http://web.mit.edu/kerberos/www/index.html). Kerberos was developed at MIT, starting around 1985 as part of project Athena. Kerberos was designed to be a network authentication service. It also provides confidentiality and integrity services to data during transit.

Kerberos works using a trusted third-party model. That is, somewhere on the network a Kerberos Authentication Server exists which can verify the identity of the client and server and provide proof to the other party of that authentication. Authentication is actually achieved by the client (acting on behalf of a user) and the server providing proof to the Kerberos Authentication Server that they know a DES cryptographic key (the user and server share their DES cryptographic key with the Kerberos Authentication Server).

Kerberos is very successful if industry acceptance is any guide. It is now being used widely for securing networks, although in many cases you may not realize you are actually using it. For example, Kerberos is built into a number of operating systems providing security to the rtools and NFS on Linux and Solaris, and it is being used as the basis of the cryptographic library on Windows NT. It is also part of a number of firewall implementations.

So What's Wrong With Kerberos?

Kerberos was designed for a closed Intranet environment, where the users and applications are managed by one administration. Although Kerberos is a good security architecture for such an isolated network, it has problems scaling to a large inter-networked environment (e.g., the Internet). Kerberos has two main problems that limit its scalability: the use of DES cryptographic keys for authentication and the lack of a user privileges (authorization) service.

The lack of scalability through the use of DES cryptographic keys for authentication is known as a key management problem. For authentication to be possible between client or server and the Kerberos Authentication Server in such an arrangement, the system administrator needs to generate a DES cryptographic key and share it securely with the two parties involved. In a small network with a single group of system administrators it is a feasible solution, but in a large inter-network, securely sharing keys is difficult.

Kerberos also lacks an authorization service. When a client application working on behalf of a user connects to the server, the server needs to know the privileges of the user. Kerberos depends on the fact that the user's privileges are already stored on the server computer. For example, the user has an

account on the server, or the user's privileges are known to the server application. This arrangement is not scalable, because in a large inter-network it is certainly not possible for every server computer to be configured with every possible user's privileges. We require another method for the server to gather the user's privileges.

SESAME

SESAME is the Secure European System For Applications in a Multi-Vendor Environment (http://www.esat.kuleuven.ac.be/cosic/sesame.html). Some people call it the European equivalent of Kerberos, but in reality it is a much improved security architecture. SESAME provides the complete authentication service of Kerberos, including confidentiality and integrity during transit, and adds to it an authorization service and public key cryptography. Both additions allow SESAME to be much more scalable to a large networked environment than Kerberos. It also adds an excellent auditing system and provides a privilege delegation scheme.

SESAME was also designed to be completely platform independent. It uses a role-based, access-control model that allows SESAME to easily interoperate with many operating systems.

SESAME development began around 1990 by ICL, Bull and Siemens Nixdorf, with the current release SESAME Version 4 (which is the version we have made available for Linux).

SESAME has an Authentication Server similar to Kerberos, but adds to it a Privilege Attribute Server. The idea here is that a user can not only be authenticated, but also provided with privileges, which can be presented to the server when required. This allows a user to access a server that has no knowledge of a user but is able to verify the privileges provided. SESAME also allows authentication based on public key technologies (as well as the standard Kerberos DES cryptographic key method).

SESAME is also gaining industry acceptance. The ICL Access Manager (http://www.icl.co.uk/access/) and ISM Access Master (http://www.ism.bull.net/) are commercial products based on SESAME. These products are being used to secure large Internet wide networks.

Securing Applications With SESAME

SESAME provides the Generic Security Services Application Programmer Interface (GSSAPI), which is a library of security routines. The aim of the library is to provide a standard method to secure client/server networked applications, and the GSSAPI is now an Internet Standard (RFC1508). Our experience with the

GSSAPI is that it is small enough to be easily understood (only about 20 routines), although it takes some time to understand all of the possibilities of each routine. The GSSAPI is becoming increasingly popular for securing applications, and the SESAME version of the GSSAPI provides the full implementation.

Figure 1 shows some segments of GSSAPI code. In the code segment, the client is authenticated to the server and data is protected during transit. The segment highlights the fact that it only takes a dozen or so extra lines of code in your client and server application to secure them (other than variable declarations). In the code segment only the client is authenticated, although with a few extra lines of code the server could also be authenticated to the client.

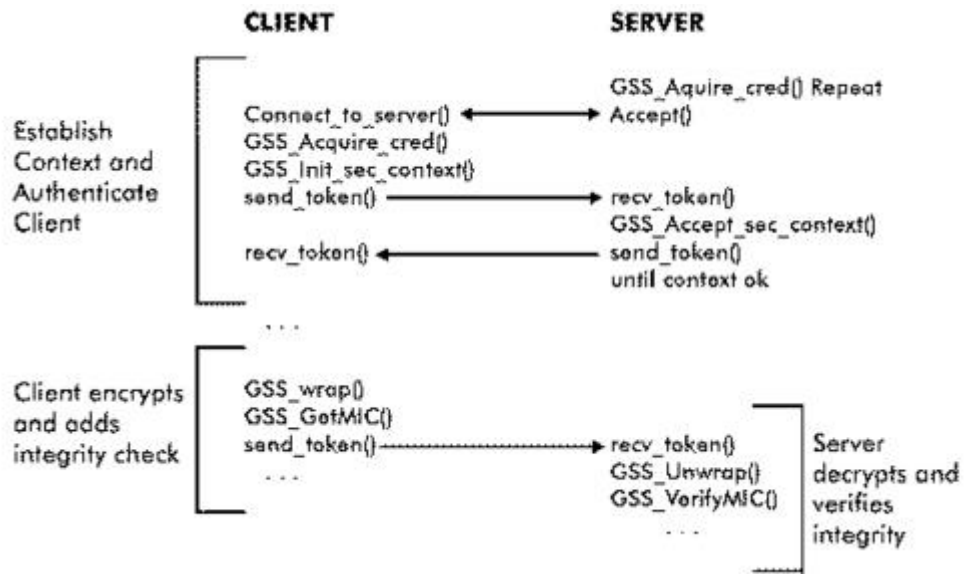


Figure 1. GSSAPI Code Representation

To secure your client/server applications, you insert the GSSAPI library calls at the appropriate points in your code and then rebuild the application. In a very short time it may be possible to convert an insecure application into a secure one, depending on how well structured your application is.

SESAME and Linux

SESAME is already available on a range of platforms: AIX 3.2 on Bull DPX 20, SINIX (Unix SVR4) on SNI MX300i, Unix SVR4 on ICL DRS6000 and AIX 3.2 on IBM RS6000. We have spent around 12 months porting SESAME to Linux. The main problems were:

- The SESAME source made numerous assumptions about the Unix environment on which it was being built. These include absolute paths in scripts for Unix programs, assuming that the root home directory was / (in our case it was /root) and so on.

- The documentation was quite extensive but still did not make it easy to build and configure the system. The order of information was not always logical, and in some sections was far too brief.
- The code had a number of memory bugs. These include over-running array bounds and memory leakages.

After securing a number of applications, we are happy with the stability of our Linux version of SESAME. It is already being used here in Australia, in Europe and in North America. We have written comprehensive building, installation and configuration guides and have a number of reports available to help you get SESAME working on your networks (http://www.fit.qut.edu.au/~ashley/sesame.html).

To get SESAME working, you first download the source from the European web site (listed in the beginning of the SESAME section) and then download our Linux patches to modify the source and build SESAME for you (we have automated it down to a one line execution). After this you follow our installation and configuration guides, which describe how to start the SESAME Security Servers, how to setup accounts for users and how to create the cryptographic keys that will be used for your security. New administrators of SESAME will probably take about two days to get SESAME working and understand what they are doing.

We are also working on building a library of SESAMIZED applications for Linux. In cooperation with other SESAME developers, we have concentrated on producing a SESAMIZED TELNET, FTP, rtools and NFS. This development is ongoing with the aim of providing a comprehensive suite of applications for Linux networks.

We have concentrated on Red Hat Linux for our port. There was no particular reason for using this version of Linux other than we are using it for related work. The first version of the port was completed on Red Hat Version 3.0.3, although lately we have it working on Red Hat Version 4.1. We have also tried SESAME on Slackware Linux and it worked without any modification.

To Sum Up

SESAME is an advanced, scalable network security architecture. SESAME's GSSAPI allows you to quickly secure your client/server applications. It provides all of the services of Kerberos, with the added advantage of being scalable as your network grows. SESAME is now available for Linux, together with comprehensive documentation, and a comprehensive suite of SESAMIZED applications for Linux is under development.

Glossary

Special Thanks

We would like to thank Shaw Innes and Mark Rutherford for their considerable efforts in chasing and fixing problems in the Linux port.

Paul Ashley is a Lecturer for the School of Data Communications, QUT. His research interests are Network Security, Health Care Security and Applications of Cryptology. He also consults for industry and government in security related projects. He can be reached using e-mail at ashley@fit.qut.edu.au or via web www.fit.qut.edu.au/~ashley/.

Bradley Broom is a Senior Lecturer for the School of Data Communications, QUT. His research interests encompass network and distributed file systems, and since moving to QUT in 1994, he has been particularly interested in the security, or lack thereof, in these systems. He can be reached using e-mail at brrom@fit.qut.edu.au or via the web at www.fit.qut.edu.au/~broom/.

Archive Index Issue Table of Contents

Advanced search

Playing with Binary Formats

Alessandro Rubini

Issue #45, January 1998

This article explains how kernel modules can add new binary formats to a system and show a pair of examples.

One of the roles that kernel modules can accomplish is adding new binary formats to a running system. A “binary format” is basically a data structure responsible for executing program files—the ones marked with execute permission. The code I'm going to introduce runs with version 2.0 of the kernel.

Kernel modules are meant to add new capabilities to a Linux system, device drivers being the best known such “capabilities”. As a matter of fact, the highly modular design of the Linux kernel allows run-time insertion of many features other than device drivers—we saw a few months ago how /proc files and **sysctl** entry points can be created by modularized code.

One other such loadable feature is the ability to execute a binary format; this includes both executable files and shared libraries. While the mechanism of loading compiled program files and shared libraries is quite elaborate, the average Linux user can easily add loaders that invoke an interpreter for new binary formats. Thus, the user is able to call data files by name and have them “executed”, after invoking **chmod +x** on the file.

How a File Gets Executed

Let's start this discussion by looking at how the **exec** system call is implemented in Linux. This is an interesting part of the kernel, as the ability to execute programs is at the basis of system operations.

The entry point of exec lives in the architecture-dependent tree of the source files, but all the interesing code is part of fs/exec.c (all pathnames here refer to /usr/src/linux/ or the location of your sources). To check architecture-specific details, locate the function by typing the command:

```
arch/*/kernel/*.c
```

Within fs/exec.c the toplevel function, **do_execve()**, is less than fifty lines of code in length. Its role is checking for errors, filling the “binary parameter” structure (**struct linux_binprm**) and looking for a binary handler. The last step is performed by **search_binary_handler()**, another function in the same file. The magic of **do_execve()** is contained in this last function which is very short. Its job consists of scanning a list of registered binary formats, passing the **binprm** structure to all of them until one succeeds. If no handler is able to deal with the executable file, the kernel tries to load a new handler via **kerneld** and scans the list once again. If no binary format is able to run the executable file, the system call returns the **ENOEXEC** error code (“Exec format error”).

The main problem with this kind of implementation is in keeping Linux compatible with the standard Unix behaviour. That is, any executable text file that begins with **#!** must be executed by the interpreter it asks for, and any other executable text is run by /bin/sh. The former issue is easily dealt with by a binary format specialized in running interpreter files (fs/binfmt_script.c), and the interpreter itself is run by calling search_binary_handler() once again. This function is designed to be reentrant, and binfmt_script checks against double invocation. The latter issue is mainly an historical relic and is simply ignored by the kernel. The program trying to execute the file takes care of it. Such a program is usually your shell or **make**. It's interesting to note that while recent versions of **gmake** execute properly when a script has no leading **#!** line, previous versions didn't call a shell resulting in a “cannot execute binary file” message when running unadorned scripts from within a Makefile.

All the magic handling of data structures needed to replace the old executable image with the new one is performed by the specific binary loader, based on utility functions exported by the kernel. If you would like to take a look at such code, the function **load_out_binary()** in fs/binfmt_aout.c is a good place to start —easier than the ELF loader.

Registration of Binary Formats

The implementation of **exec** is interesting code, but Linux has more to offer: registration of new binary formats at run time. The implementation is quite straightforward, although it involves mucking with rather elaborate data structures—either the code or the data structures must accomodate the underlying complexities; elaborate data structures offer more flexibility than elaborate code.

The core of a binary format is represented in the kernel by a structure called **struct<\!s>linux_binfmt**, which is declared in the linux/binfmts.h file as follows:

```
struct linux_binfmt {
        struct linux_binfmt *next;
        long *use_count;
        int (*load_binary)(struct linux_binprm *,
                struct pt_regs *);
        int (*load_shlib)(int fd);
        int (*core_dump)(long signr,
                struct pt_regs *);
        };
```

The three functions, or “methods”, declared by the binary format are used to execute a program file, to load a shared library and to create a core file. The **next** pointer is used by search_binary_handler(), while the **use_count** pointer keeps track of the usage count of modules. Whenever a process **p** is executing in the realm of a modularized binary format, the kernel keeps track of ***(p->binfmt->use_count)** to prevent unexpected removal of the module.

A module, then, uses the following functions to load and unload itself:

```
extern int register_binfmt(struct linux_binfmt *);
extern int unregister_binfmt(struct linux_binfmt *);
```

The functions receive a single argument instead of the usual pointer,name pair because no file in the /proc directory lists the available binary formats. The typical code for loading and unloading a binary format, therefore, is as simple as the following:

```
int init_module (void) {
  return register_binfmt(&bluff_format);
}
void cleanup_module(void) {
  unregister_binfmt(&bluff_format);
}
```

The previous lines belong to the **bluff** module (Binary Loader for an Ultimately Fallacious Format), whose source is available for public download from ftp://ftp.linuxjournal.com/pub/lj/listings/issue45/2568.tgz.

The structure representing the binary format can declare as NULL any of the functions it offers; NULL functions are simply ignored by the kernel. The easiest binary format, therefore, looks like the following, which is the one used by the bluff module:

```
struct linux_binfmt bluff_format = {
        NULL, &mod_use_count_, /* next, count */
        NULL, NULL, NULL    /* bin, lib, core */
};
```

Yes, **bluff** *is* a bluff; you can load and unload it at will, but it does absolutely nothing.

The Binary Parameters

In order to implement a binary format that is of some use, the programmer must have some background information about the arguments that are passed to the loading function, i.e., **format->load_binary**. The first such argument contains a description of the binary file and the parameters, and the second is a pointer to the processor registers.

The second argument is only needed by *real* binary loaders, like the a.out and ELF formats that are part of the Linux kernel sources. When the kernel replaces an executable file with a new one, it must initialize the registers associated with the current process to a sane state. In particular, the instruction pointer must be set to the address where execution of the new program must begin. The function **start_thread** is exported by the kernel to ease setting up the instruction pointer. In this article I won't go so deep as to describe real loaders but will limit the discussion to “wrapper” binary formats, similar to binfmt_script and binfmt_java.

The **linux_binprm** structure, on the other hand, must be used even by simple loaders, so it is worth describing here. The structure contains the following fields:

- **char buf[128]**: This buffer holds the first bytes of the executable image. It is usually looked up by each binary format in order to detect the file type. If you are curious about the known magic numbers used to detect the different file formats, you can look in the text file /usr/lib/magic (sometimes called /etc/magic).
- **unsigned long page[MAX_ARG_PAGES]**: This array holds the addresses of data pages used to carry around the environment and the argument list for the new program. The pages are only allocated when they are used; no memory is wasted when the environment and argument lists are small. The macro **MAX_ARG_PAGES** is declared in the binfmts.h header and is currently set to 32 (128KB, 256KB on the Alpha). If you get the message “**Arg list too long**” when trying to run a massive grep, then you need to enlarge **MAX_ARG_PAGES**.
- **unsigned long p**: This is a “pointer” to data kept in the pages just described. Data is pushed to the pages from high addresses to low ones, and **p** always points to the beginning of such data. Binary formats can use the pointer to play with the initial arguments that are passed to the program being executed, and I'll show such use in the next section. It's interesting to note that **p** is a pointer to user-space addresses, and it is expressed as **unsigned long** to avoid an undesired de-reference of its value. When an address represents generic data (or an offset in the memory “array”) the kernel often considers it a long integer.

- **struct inode *inode**: This inode represents the file being executed.
- **int e_uid, e_gid**: These fields are the effective user and group ID of the process executing the program. If the program is **set-uid**, these fields represent the new values.
- **int argc, envc**: These values represent the number of arguments passed to the new program and the number of environment variables.
- **char *filename**: This is the full pathname of the program being executed. This string lives in kernel space and is the first argument received by the **execve** system call. Although the user program won't know its full pathname, the information is available to the binary formats, so they can play games with the argument list.
- **int dont_iput**: This flag can be set by the binary format to tell the upper layer that the inode has already been released by the loader.

The structure also contains other fields that are not relevant to the implementation of simple binary formats. What is relevant, on the other hand, are a pair of functions exported by exec.c. The functions are meant to help the job of simple binary loaders such as the ones I'll introduce in this article.

```
unsigned long copy_strings(int argc,char ** argv,
        unsigned long *page, unsigned long p,
        int from_kmem);
void remove_arg_zero(struct linux_binprm *bprm);
```

The first function is in charge of copying **argc** strings, from the array **argv** into the pointer **p** (a user space pointer, usually **bprm->p**). The strings will be copied before the address pointed to by **p** (argument strings grow downwards). The original strings, the ones in **argv**, can either reside in user space or in kernel space, and the array can be in kernel space even if the strings are stored in user space. The **from_kmem** argument is used to specify whether the original strings and array are both in user space (0), both in kernel space (2) or the strings are in user space and the array in kernel space (1). **remove_arg_zero** removes the first argument from **bprm** by incrementing **bprm->p**.

A Sample Implementation: Displaying Images

To turn the theory into sound practice, let's try to expand our **bluff** into **bloom** (Binary Loader for Outrageously Ostentatious Modules). The complete source of the new module is distributed together with bluff.

The role of bloom is to display executable images. Give execution permission to your GIF images and load the module, then call your image like it was a command, and **xv** will display it.

This code is neither particularly original (most of it comes from binfmt_script.c) nor particularly smart (text-only people like me would rather use an ASCII

viewer, for instance, and other people prefer a different viewer). I feel this kind of example is quite didactic anyway, and it can be easily run by anyone who can run an X server and has root access to the computer in order to load modules.

The source file is made up of little more than 50 lines and is able to execute GIF, TIFF and the various PBM formats; needless to say, you must give your images execute permissions (**chmod +x**) in advance. The viewer is configurable at load time and defaults to /usr/X11R6/bin/xv. Here is a sample session copied from my text console:

```
# insmod bloom.o
# ./snowy.tif
xv: Can't open display
# rmmod bloom
# insmod bloom.o viewer="/bin/cat"
# ./snowy.tif | wc -c
1067564
```

If you use the default viewer and work within a graphic session, your image file will bloom on the display.

If you can't wait to download the source file, you can see the interesting part of bloom in Listing 1. Note that **bloom.c** falls under the GPL, because most of its code is copied from **binfmt_script.c**.

Listing 1. The Core of bloom.c

Registering the Format with kerneld

The next question that I hear you ask is “How can I set up things so that **kerneld** can automatically load my module?”

Well, actually it isn't always possible. The code in fs/exec.c only tries to use kerneld when at least one of the first four bytes is not printable. This behaviour is meant to avoid losing too much time with kerneld when the file being executed is a text file without the **#!** line. While real binary formats have one non-printable byte in the first four, this isn't always true for generic data types.

The net result of this behaviour is that you can't automatically load the bloom viewer when invoking a GIF file or when calling a PBM file by name. Both formats begin with a text string and will therefore be ignored by the auto-loader.

When, on the other hand, the file has a non-printing character within the first four, the kernel issues a kerneld request for binfmt-number, where the exact string is generated by this statement:

```
sprintf(modname, "binfmt-%hd",
        *(short*)(&bprm->buf));
```

The ID of the binary format generated by the above statement represents the first two bytes of the disk file. If you try to execute TIFF files, kerneld looks for **binfmt-19789** or **binfmt-18761**. A gzipped file calls for **binfmt--29921** (negative). GIF files, on the other hand, are passed to /bin/sh shell due to their leading text string. If you want to know the number associated with each binary format, look in the /usr/lib/magic file and convert the values to decimal. Alternatively, you can pass the debug argument to kerneld and look at its messages when you execute your data files and it tries to load the corresponding binary format.

It's interesting to note that kernel versions 2.1.23 and newer switched to an easier and more significant ID by using the following line:

```
sprintf(modname, "binfmt-%04x",
               *(unsigned short *)(&bprm->buf[2]));
```

This new ID string represents the third and fourth byte of the binary file and is hexadecimal instead of decimal (thus leading to strings with a better format and no ugly “minus-minus” appearing now and then.

What's This Worth?

While calling images by name can be funny, it has no real role in a computer system. I personally prefer calling my viewer by name, and I do not believe in the object-orientedness of the approach. This kind of feature in my opinion is best suited to the file manager where it can be tailored by appropriate configuration files without introducing kernel bloat to lie in the way of any computational path.

What *is* really interesting about binary formats is the ability to run program files that don't fall in the handy **#!** notation. This includes executable files belonging to other operating systems or platforms, as well as interpreted languages that have not been designed for the Unix operating system—all those languages that complain about a **#!** in the first line.

If you want to play one such game, you can try the **fail** module. This “Format for Automatically Interpreting Lisp” is a wrapper to invoke Emacs any time a byte-compiled e-lisp program is invoked by name. Such practice is definitely failure-prone, as it makes little sense to invoke several megabytes of program code to run a few lines of lisp. Moreover, Emacs-lisp is not suited to command-line handling. Together with **fail** you'll also find a pair of sample lisp executables to make your tests.

A real-world Linux system is full of interesting examples of interpreted binary formats such as the Java binary format. Other examples are the binary format that allows the Alpha platform to run Linux-x86 binaries and the one included in recent DOSEMU distributions that is able to run old DOS programs transparently (although the program must be specifically tailored in advance).

Version 2.1.43 of the kernel and newer ones include generic support for interpreted binary formats. **binfmt_misc** is somewhat like bloom but much more powerful. You can add new interpreted binary formats to the module by writing the relevant information to the file /proc/sys/fs/binfmt_misc.

Listing 1 and all other programs referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue45/2568.tgz.

Alessandro Rubini used to read e-mail at his university account, but then abandoned academia because he was forced to write articles. He now reads e-mail as rubini@linux.it and still writes articles.

Archive Index Issue Table of Contents

Advanced search

Extra

Amy Kukuk

Issue #45, January 1998

Geek vocab.

What's It?

burn-in: One of the quality tests performed on electrical circuits in computer equipment during the manufacturing process. During the burn-in process, the temperature may be varied from below freezing to above 100 degrees Fahrenheit to test the circuits in a computer or its components while they are operating. In some tests, the input voltage may be varied.

latency: Delay between when a computer receives an address to which data is to be transferred and when it actually starts the transfer.

message-passing: Term related to distributed multiprocessing operating systems for communications between tasks.

MIMD: Multiple instructions, Multiple Data machine. Massive parallel processing architecture in which the processors work as a team, solving large problems by dividing them up. Each processor has its own memory. The number of processors in a MIMD system varies from 16 to 2000. Each processor manipulates different data independently.

parallel programming: Writing a program so that separate elements of it are executed at the same time. Concurrent C/C++ is an example of a language written for parallel programming.

PCI bus: Peripheral component interconnect bus. The local bus standard developed by Intel Corp. which allows the central processing unit to transfer data to 16 devices at 33MHz along a 32- or 64-bit pathway. This version is a separate bus isolated from the CPU.

RS-232: Standard for cable and 25-pin electrical connection between computers and peripheral devices using a serial binary data interchange. Used for slower communications, requiring speeds of no greater that 20Kbps, with a standard limit of 75 feet.

SIMD: Single instruction, multiple data. Massively parallel processing architecture with large numbers of processors working on a single problem but sharing distributed memory. SIMD computers have between 1000 and 16,400 processors.

virtual: Anything that appears to be other than what it actually is, e.g., virtual memory is the apparent expansion of the computer's memory by using disk space to store programs and data.

Archive Index Issue Table of Contents

Advanced search

Remote Compilation Using ssh and make

John R. Daily

Issue #45, January 1998

Here's a quick lesson in setting up scripts to use the ssh and make commands for compiling on a remote machine.

Occasionally I use my Linux machine at home to write code that I intend to compile on a remote machine.

While maintaining open FTP and TELNET connections to the remote machine to handle the transfer and compilation steps is a manageable solution, I decided to explore **ssh** and **make** to develop a more automated method.

The benefits of my solution:

- No need to remember which files have been modified.
- Ability to use Emacs' compilation capabilities to move to errors in the source.
- As mentioned above, no need to use FTP or TELNET, and hence no reason to keep an open dial-up connection when not compiling.
- Automate, automate, automate—laziness is a virtue.

Overview of Solution

My first step was to set up ssh and related tools in order to allow secure copying of files to my remote system. While setting up a .rhosts file is a (barely) acceptable solution, my IP address and name is different each time I dial in, and it would be rather awkward to change the remote system's .rhosts file each time I dialed in.

ssh gives me a much more secure form of authentication for copying files and executing remote commands.

Once I had ssh behaving properly, I used Emacs' **info** facility to explore implicit rules in Makefiles and wrote a simple Makefile to handle the file transfers and remote compilation.

As an example of the intended effect, assume my remote machine is called "remote" (and my local machine "local"), and I've just modified a source file called daemon.c. I would like to execute the following commands in an automated fashion (note that **scp** is a secure copy command packaged with ssh, and that the **-C** option specifies compression, useful for dialup connections):

```
scp -C daemon.c jdaily@remote:~/source-directory
ssh -C remote "cd source-directory && make"
```

Implementation

First, I needed **sshd** running on the remote system to handle my secure connections. Fortunately, sshd was already running on the remote system in question. According to the man pages, it can be run by any user (i.e., user does not have to be root) and is restricted to handling connections for that user (which should provide sufficient security for our needs).

Then, I needed to install the ssh tool set on my local machine. Again, ideally these would be installed in a public binary directory such as /usr/local/bin, but any user can install them in his/her home directory.

I also wanted a key which would allow me to authenticate myself between systems and eliminate the need to type my password each time I tried to run one of the ssh commands. For this, I ran ssh-keygen and made sure not to give a pass phrase, so that none would be needed when using my private key to establish the connection. Listing 1 shows an example ssh-keygen session.

Once I had a public key, I used scp to copy it to the remote machine.

$ scp -C ~/.ssh/identity.pub \jdaily@remote:~/.ssh/keyjdaily's password:
<entered my remote password>

Then I logged onto the remote host and copied the key file into ~/.ssh/authorized_keys. If that file had already existed, I would have appended the key file.

Following all this, I could run ssh and scp without needing either a password or a pass phrase to connect to remote.

Now I needed a makefile to automate my system. Ideally, the files on the remote machine would be checked to see if they were older than the files on

my local machine, and if so, they would be replaced. To simplify matters, I decided to keep a record of the “last transferred date” for each file by touching a corresponding file each time I copied a source file over.

As an example, when I transferred a newer copy of daemon.c over, I touched daemon.ct in the same directory. Any transfer of a .h file would be marked by the creation of a file with a .ht suffix.

After poking around the info file for make, I came up with the Makefile shown in Listing 2. This Makefile had one limitation in particular; I was unable to specify command-line arguments for make on the remote machine without writing them directly into the makefile on my local system. While this was fine for the current application, I decided to generalize it by creating a **run-make** shell script which would handle the remote execution of make after calling make on the local system. Listing 3 is my run-make shell script. I then removed the line from my Makefile which remotely ran make. Listing 4 is the output from a successful compilation sequence.

Resources

John Daily works for BBN (or is that GTE?) as a systems administrator/software engineer. He spends far too much time in front of computers. Whenever he can, he prefers to be outside riding his new bicycle and exploring New England. He can be reached via e-mail at jdaily@bbn.com.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #45, January 1998

Our experts answer your technical questions.

Touch Screen Support

I am currently looking into the possibility of using a touch screen for my system. Which touch screens are supported by Linux? —Cheng Yap Red Hat 4.2

Touch screen support depends on the X server you are using. I'm not sure which (if any) touch screens are supported by XFree86. If a commercial server is an option, Metro-X has support for some touch screens. You can get more information about it at http://www.metrolink.com/. —Steven Pritchard, President Southern Illinois Linux Users Group steve@silug.org

There is not yet a standard for touch screens. Each manufacturer has different requirements and features. They also have different methods of communicating “touch” information to the PC.

Your best bet is to find a touch screen that communicates using one of the common mouse protocols. You then hook the screen (or cover that slips onto a screen) to a communications port on your PC and install **GPM** or tell X to use that port as its mouse port. —Chad Robinson, BRT Technologies Senior Systems Analyst chadr@brt.com

Linking Linux and Windows NT

I have a Linux machine on the same network as a few Windows NT 4.0 machines. I'd like to convert more machines to Linux boxes but can't, at this time, so I want to share resources between these machines. How can I easily share files between Linux and Windows NT machines? Also, can I share printers?

Ideally, I'd like to get the Linux machine to act as a file server for the NT workstations, since I didn't shell out the extra for NT Server. Is this possible? —Alex Tan Red Hat 4.2

SAMBA can give you the exact functionality you describe in your e-mail. Using SAMBA you can configure your Linux box to act both as a file and printer server for your NT (or Windows 95) machines. For more information about installing and configuring SAMBA check the SMB-HOWTO in your /usr/doc/HOWTO directory. —Mario de Mello Bittencourt Neto, WebSlave System Administrator mneto@buriti.com.br

Installing Perl

I recently purchased Red Hat Linux 4.1 and I want to install Perl. I have no idea of how to even begin. I've been told that I need to download it first. Do I? From which site? Is there documentation on step by step installation? —Enrique Estrada Red Hat 4.1

You're in luck, because Perl is already a part of Red Hat 4.1 . If you did a standard installation it is probably already installed. To confirm this, at the shell prompt type:

```
perl -v
```

If perl is installed, it will report back the current version number. If it was missed during the installation process, you'll need to read up on using the Red Hat Package Manager (RPM) to install it from the disks.

If you want to download the latest versions you can pick them up from many different sites. The web pages at http://www.perl.com/ are a great starting point. —Vince Waldon vwaldon@redcross.ca

Parallel Tape Device Support

I am searching for parallel port drivers for my employer, and I have come across some parallel port drivers for tape drives such as the HP Colorado Trakker 250. Unfortunately, this is not the one that I need. Have there been any recent developments in parallel port drivers for Linux and, if so, what projects are complete, underway or planned for the future? —Scott Mulroy

A lot of pointers are available on the Linux Parallel Port home page at URL http://www.torque.net/linux-pp.html/. —Pierre Ficheux, Lectra Systèmès pierre@rd.lectra.fr

Sound Drivers for S.u.S.E

I'm very satisfied with my S.u.S.E. Linux distribution, but I'm not able to make my Soundblaster 16 PnP (plug and play) work. Do you have any suggestions? —Stefan Stahl S.u.S.E. 5.0

The best solution for me was to use **Pnp4Kernel**. You can get it from the Red Hat Linux-PnP web page (http://www.redhat.com/linux-info/pnp/).

Another easy solution is to buy the commercial sound driver by 4 Front Technologies (OSS). It's cheap ($20), easy to install and works well. Check out http://www.4front-tech.com/linux/. —Pierre Ficheux, Lectra Systèmès pierre@rd.lectra.fr

Windows Only Printer?

Is it possible to get a laser printer that only supports Windows 3.1, 95 or NT working under Linux? All drivers are normally installed in the Windows directory and the printer is not able do anything without Windows. —Jan Rooijackers InfoMagic April 1997

That depends entirely upon your printer's facilities. First, run a quick test from MS-DOS. Click Start, Shut Down and then Restart in MS-DOS mode. At the command prompt, type:

```
copy c:\autoexec.bat lpt1
```

*replacing **lpt1** with your printer. If your printer produces a printout of your autoexec.bat file, you know your printer does not require special software to function, and Linux should be able to operate the printer without difficulty. If it does not, it's possible that the manufacturer (to save money) has removed functionality from the printer itself and placed it in the driver, in the same manner that the "WinModem" was created.*

If you are able to print without the driver, you should then look to see what command set your printer emulates. The industry standard is Hewlett Packard's PCL. Many printers directly support this standard or can be configured to support it. —Chad Robinson, BRT Technologies Senior Systems Analyst chadr@brt.com

Archive Index Issue Table of Contents

Advanced search