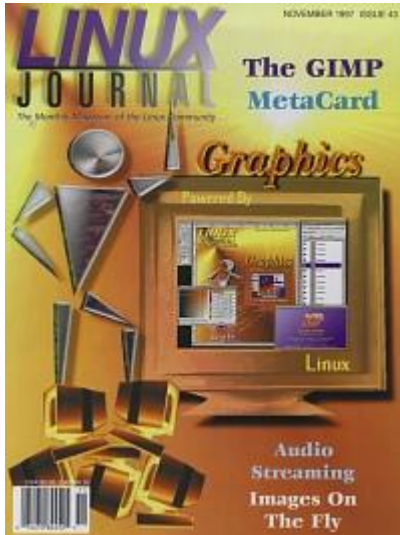


Advanced search

Linux Journal Issue #43/November 1997



Features

Graphical Applications Using MetaCard by *Scott Raney*

How to write graphical applications using MetaTalk, a MetaCard scripting language.

Streaming Audio in Linux by *Siome Goldenstein*

The nature and perception of sound, sound as an object in a computer and available software for streaming audio.

The Quick Start Guide to the GIMP, Part 1 by *Michael J. Hammel*

First of a four-part series introducing the GIMP, a Linux power tool for the graphics artist.

GIF Images On The Fly by *Jimmy Ball*

New images from old, in real time.

News & Articles

Linux Expo at Union Bank of Switzerland by *Martin Sjoelin*

Sound through the PC-speaker by *Paul Dunne*

YODL or Yet Oneother Document Language by *Karel Kubat*

Linux Journal Interviews Robert Nation by *Larry Ayers*

Linux as a Telephony Platform by *David Sugar*

Game Control Design by *Dave Thomson*

Linux and the Alpha, How to Make Your Applications Fly, Part 2 by

David Mosberger

Mistaken Identities

Reviews

- Product Review** [Metro-X 3.1.5](#) by *Mark Nassal*
Product Review [SCO OpenServer](#) by *Ken Collins*
Product Review [Lifebook 420D Notebook Computer](#) by *Michael Scott Shappe*
Product Review [AcceleratedX CDE and Display Server for PC Unix](#) by *Bradley Willson*
Book Review [Operating Systems: Design and Implementation, 2nd edition](#) by *Boycho Peytchev*
Book Review [Perl5](#) by *Michael Hines*
Book Review [Essential Linux](#) by *Marjorie Richardson*
Book Review [Linux Universe](#) by *Jan Rooijackers*

WWWsmith

- [Faxing From a Web Page](#) by *David Weis*
Book Review [The UNIX Web Server Book](#) by *Gerald Luther Graef*
At the Forge [Integrating SQL with CGI, Part 2](#) by *Reuven Lerner*

Columns

Letters to the Editor

- From the Publisher [Linux in the Mainstream?](#) by *Phil Hughes*
Stop the Presses [Ownership of Linux Trademark Resolved](#) by *Margie Richardson*
Linux Apprentice [Power Printing with MagicFilter](#) by *Bill Cunningham*
Take Command [ssh: Secure Shell](#) by *Alessandro Rubini*
Linux Means Business [Highway POS System](#) by *Marc L Allen*
New Products
System Administration [IP Masquerading Code Follow-Up](#) by *Chris Kostick*
Kernel Korner [Linux Kernel Installation](#) by *David A. Bandel*
Linux Gazette [Using SAMBA to Mount Windows 95](#) by *Jonathon Stroud*
Best of Technical Support

Archive Index

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Graphical Applications Using MetaCard

Scott Raney

Issue #43, November 1997

This article demonstrates how to write graphical applications using a MetaCard scripting language called MetaTalk.

A **top** program is a very useful thing to have on a multi-tasking operating system like Linux. You can use it to keep track of the CPU and memory usage of all programs running on the system and to detect and kill runaway processes. The character-based **top** program that comes with Linux systems can be improved. Because it isn't aware of the windowing system, it doesn't go to sleep when the window it is running in is iconified. It would also be nice if you could select a process to kill by clicking on it instead of having **top** type in its process ID.

Fortunately, Linux has a very simple and elegant way to get process information, and so it's easy to develop a graphical application to display this information using a tool like MetaCard. This article will show you how it's done using MetaCard's scripting language MetaTalk. MetaTalk is a very high level language (VHLL) that supports building complete applications with very little effort. The MetaTalk language has an English-like syntax and is very concise. This makes it easy to learn, yet doesn't sacrifice the power and compactness found in other HVLLs such as Perl. It is very readable, so it's easy for non-expert users to figure out what a script does (which can be much more difficult with languages like Perl).

The `/proc` Pseudo File System

Each process that is running on a Linux system has a directory in `/proc`. In that directory are several files containing information on the process. To implement a graphical **top** program, we're most interested in the information in the file named *stat*, which contains process run time and memory usage information. We'll also use the file named *cmdline* that contains the command line used to start up that process.

The stat file contains a single line of information with the fields separated by spaces. Details on the format can be found in the proc man page. Information on this page indicates that on a Linux 1.2 system, words 14 and 15 are the user and system run times, respectively, for a particular process. Word 23 is the process size in bytes, and word 24 is the number of 4KB pages currently in RAM for that process.

There are also files in the /proc system that contain information about the whole system. We'll use the /proc/stat file which contains overall system resource usage. We'll need that information to compute the percentage of CPU usage for each process.

The procedure

Each time the display is to be updated, a program must do the following:

1. Read the stat file in the /proc directory.
2. Find all of the subdirectories in the /proc directory.
3. Read the stat and cmdline files in each of these directories.
4. Compute the CPU time for the process by subtracting from the last time.
5. Save the current CPU usage.
6. Convert the CPU usage into a percentage of total usage.
7. Build the list of processes and display it.
8. Schedule a time to redo the update.

The MetaTalk handler that does all of these things is called **updatelist** and is shown in [Listing 1](#). This handler, like all MetaTalk message handlers, starts with the word **on** and the name of the message to be handled. The first few lines of this handler declare all the local variables used in this handler. While not strictly necessary, it's a good idea to declare variables to avoid bugs caused by misspelling variable names. To check your scripts, you can set the MetaCard property **explicitVariables**, which will flag as an error any variable used before it was declared.

The handler then gets the global system time statistics from the file /proc/stat and subtracts the values from the last time the handler was called. The time statistics are then stored in a local variable declared outside the handler, which works like a "static" variable in C. That is, it retains its value like a global, but can only be referred to within the script, so it doesn't pollute the global name space. This variable, like all MetaTalk variables, can be used as an associative array without special declarations. All you have to do is put an alphanumeric string between the [] (square brackets).

Note the expressions of the form “**word x of y**”. These are called “chunk” expressions and are a very powerful feature of MetaTalk. With them you can access elements of a string individually without having to split up the whole string into an array first. Also note that you can add words together without having to explicitly convert them into numbers first. This saves development time and makes scripts smaller than they would be in lower-level languages.

The readfile function used in the updatelist handler is shown in [Listing 2](#). Like all function handlers, it starts with the keyword **function** and is used in expressions just like MetaCard built-in functions.

There is one unusual thing about this function. Normally in MetaCard, one would use:

```
read from file x until eof
```

because when you specify **eof** as the terminating condition, MetaCard speeds things up by getting the file size and reading the whole file with a single system call. This doesn't work for /proc files since the stat() system call returns the file lengths as 0 bytes, even though they contain data. Therefore, we must force MetaCard to read the file a byte at a time by specifying **empty** as the terminating condition.

The readfile function also has to handle the case where the file specified doesn't exist, which can happen if the process exits after the **ls** command that obtains the directory contents is executed. It also needs to convert nulls (ASCII 0 characters) in the strings to spaces, since some of the files use this character as a delimiter. Note that this requires a scripting language that can handle binary data. MetaTalk has no problem with this, but many other scripting languages do.

The rest of the updatelist handler does steps 2 through 7 in the above recipe. The resulting listing is sorted twice to remedy one of the more annoying characteristics of the character-based **top** program: the individual processes bounce around the listing unpredictably if they're not using any CPU time. Instead, we'll sort them by process size and then by CPU time which is a more useful way to do it. This is only possible because MetaCard's sort is stable, which means order is preserved on sequential sorts if elements have the same key value.

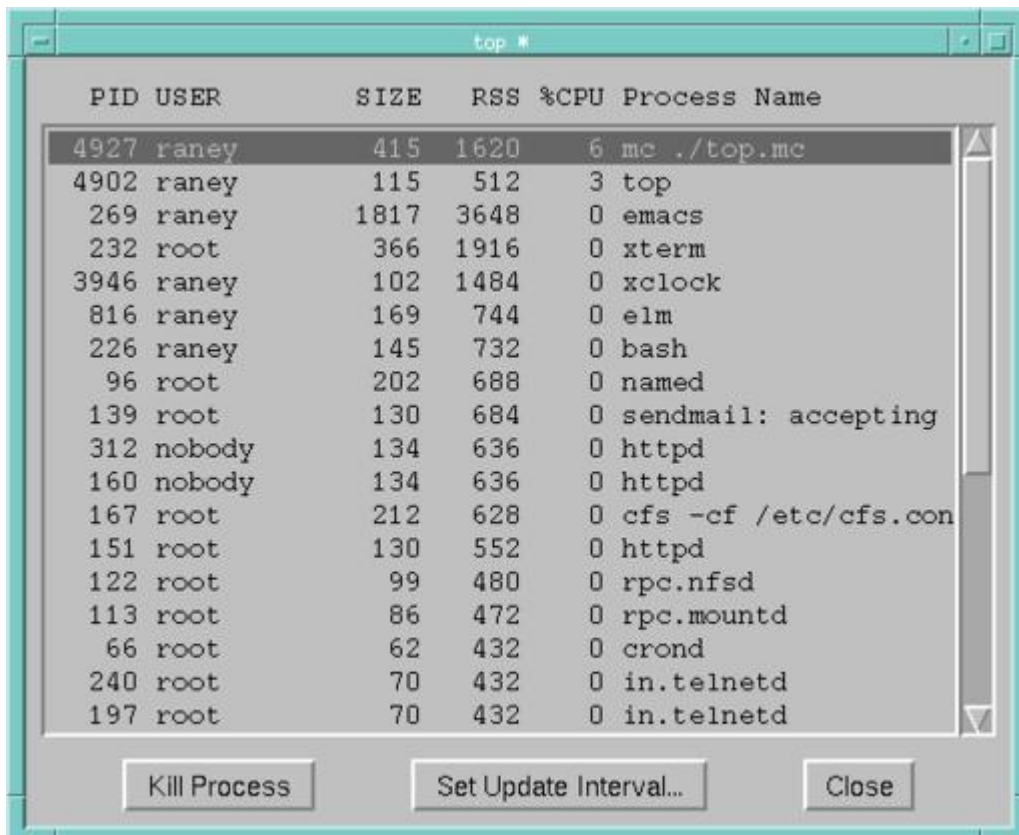
Finally, the updatelist handler uses the **send** command to schedule a call back to itself in a few seconds and stores the ID of the timer **send** creates in a variable. This local variable can be used to cancel the timer using the **cancel** function. For example, when the application is iconified, we want to stop the processing (See Listing 3).

I snuck in a bit of object-oriented programming in the **send** command. The message, sent to “me”, is the object whose script is currently executing. The time interval is specified as “**the update interval of me**”. This object has a custom property named **updateinterval** that is persistent, which means it's saved whenever the object is saved. That's why you don't see any initialization of this property in these listings; it's data stored with the stack, not the code.

Building a stack

Backtracking a little, before you can write a script, you need to create an object to attach the script to. MetaCard applications are composed of one or more stacks, each of which has one or more cards. The metaphor (inherited from HyperCard) is of a stack of index cards, but you can also think of it as being pages in a book, slides in a presentation or the frames in a movie or video. The stacks are stored as binary files similar to the resource files used on the Mac and Windows. But in addition to object descriptions, data such as the state of buttons and the text in fields is also stored in these files.

The objects are created, sized and positioned using MetaCard's IDE (Integrated Development Environment). To start building a new application window, choose “New Stack” from the File menu (the first card is created automatically with the stack). Then choose tools from the tool bar and draw the fields and buttons you need. See Figure 1 for a screen snapshot of the MetaCard **top** application.



After creating the stack and the controls in it, you can add scripts to the stack, cards and/or controls. Normally each object has its own script, but I've put all of the handlers for the **top** program in the card script. This means that the `mouseUp` handler (see [Listing 3](#)) is a little unusual, since it gets called through the message-passing hierarchy: messages not handled by an object can be handled by objects higher up in the hierarchy. In this case, a handler in the card script handles messages sent to any of the controls on the card.

Writing one big handler instead of a bunch of small ones usually means you have to figure out which object the message was originally sent to. The **target** function supplies this information. The target function returns an object that you can get the properties of or send messages to. This **mouseUp** handler also shows off the MetaCard **switch** control structure:

- The “Kill Process” case is executed when the user clicks on the “Kill Process” button. This section gets the PID of the process from the selected line in field and uses the MetaTalk **kill** command to kill it.
- The “Set Update Interval...” section prompts the user for the new *updateinterval* value using the **ask** dialog, verifies that the value is a number and tells the user that it must be if it isn't. The **ask** and **answer** dialogs are built into MetaCard and are quick and easy ways to get simple responses from the user. If the new data checks out OK, the handler cancels the current timer and then calls the `updatelist` handler to restart it.
- The “toplist” case will be executed when the user clicks on a line in the main field. This case enables the “Kill Process” button and suppresses updates for 5 seconds, allowing the user time to kill the process before the selection is cleared when the field is next updated.

[Listing 4](#) shows the handlers for the stack-oriented messages, including those sent when the stack opens and closes and is iconified and uniconfied. Remember the goal of having the application go to sleep when it is iconified.

MetaCard doesn't have a constraint-based geometry system, so you must write scripts to handle resizing and repositioning controls when a stack window is resized. The **resizeStack** message handler that does this geometry management is shown in Listing 2. Using the IDE, I set the stack properties such that the stack is not resizable in width, only in height (since you need to be able to see all fields). So this handler only has to resize the main field vertically and reposition the buttons along the bottom edge of the display. This simple four-statement handler reliably handles the task without triggering the time-consuming trial-and-error phase of development required to get a constraint-based system working correctly.

Performance characteristics

Running the MetaCard version of **top** takes about twice as much CPU time as the character-based version (6% vs. 3% on a Pentium 90 running Linux 1.2.13). This is a typical result, since a well-written MetaCard script generally runs 2 to 4 times slower than a comparable C program. Of course, the MetaCard version only took a fraction of time to develop. And because it is so much smaller, it will take far less time and effort to maintain and customize. The character-based **top** program is written in C and is about 10 times as long. Memory usage for the MetaCard version of **top** is considerably less than the total of the character-based **top** program added to the memory needed for the xterm and the extra bash process required to run it.

Conclusion

The real power of this graphical **top** is that it's easy to modify to suit your needs. You could easily add columns to display some of the other information shown in the character-based **top**, or change the signal used to kill a process. Since MetaCard has built-in object graphics capabilities, you could even add a graphical display of the resources used by individual processes over time.

You can get the MetaCard **top** stack from *Linux Journal's* [ftp site](#), but you'll need to get the MetaCard engine from the MetaCard WWW site <http://www.metacard.com> or FTP site <ftp://ftp.metacard.com/MetaCard/>.

Scott Raney (raney@metacard.com) is president of MetaCard Corporation, a vendor of multimedia and GUI development tools based on high-level languages and direct manipulation interfaces.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Streaming Audio in Linux

Siome Goldenstein

Issue #43, November 1997

This article presents a brief description of the nature and perception of sound, how to deal with sound as an object in a computer and the available software for streaming audio.

With the recent popularization of multimedia and the ever decreasing cost of hardware, the use of sound by developers in new applications is becoming more appealing each day. Sound can flavor a game, help in the user interface of a word processor, and more, making using a computer more fun and/or more effective.

There are two different forms of sound generation in the common sound boards. The first one is used primarily for music, through a MIDI interface and FM generators. The idea is to synthesize, artificially or by some previous recording samples, the occurrence of different instruments at different times. The second approach is to use digital audio signals, which can represent any sound, from a bird chirp to a human voice—even musical signals.

Sound and Perception

The human auditory system perceives differences in the pressure of the air that reaches it. When we pluck a guitar string, for example, it starts to vibrate. This movement propagates through the air in waves of compression and decompression, just as we can see waves over a lake's surface when we throw a small rock into it.

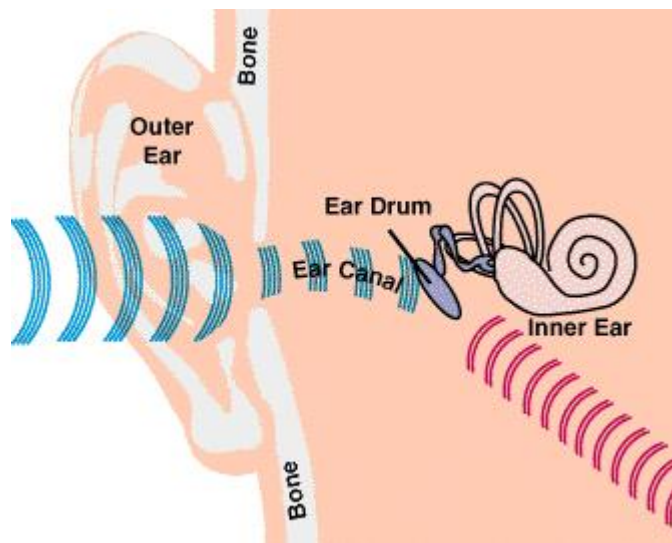


Figure 1. Human Auditory System

Figure 1 shows a detailed representation of the human auditory system, which processes and decomposes the incoming sound before sending the information to the brain. There is a lot of literature about processing human speech (see References 1 and 2), however, I'll give just a little of the basics rather than going into too much detail about the bio-medical considerations.

We “understand” fast oscillations of the air as sharp sounds and slow oscillations as bass sounds. When different oscillations are mixed together, we may lose this “musical” connotation. It is hard to classify the sound of a gun or the crash of a glass. Nevertheless, there are some mathematical tools able to decompose an arbitrary sound into a set of simple oscillations (e.g., Fourier Analysis), but these tools are quite beyond the scope of this article (see Reference 3).

A simple mathematical method for the representation of a sound phenomenon is the use of a function that describes it. This function is responsible for relating the intensity of the air pressure over a period of time. This approach is commonly called *temporal representation* and is also obtained when using electronic transducers (like microphones), which map electric voltages instead of air pressures over time. At the end of electronic processing, such as recording or amplifying, another transducer (a speaker, for example) is responsible for obtaining the air pressures from the electric intensities so that we can hear them.

For analog electronics this model is good enough; however, for the use of modern digital computers or DSPs (special devices for signal processing), there is one problem—how to keep and process the infinite information, contained in even a small interval of a function, in a limited amount of memory.

We have to work with digital audio, obtained from the continuous mathematic functional model, through two steps: sampling and quantization. The sampling step represents the domain of the function with only a limited number of points. This is usually done through *uniform punctual sampling*; in other words, we keep the value of the function only at some points, equally spaced from each other. It is clear that some loss of information may occur if by any chance the function varies too much between two adjacent samples (see Figure 2).

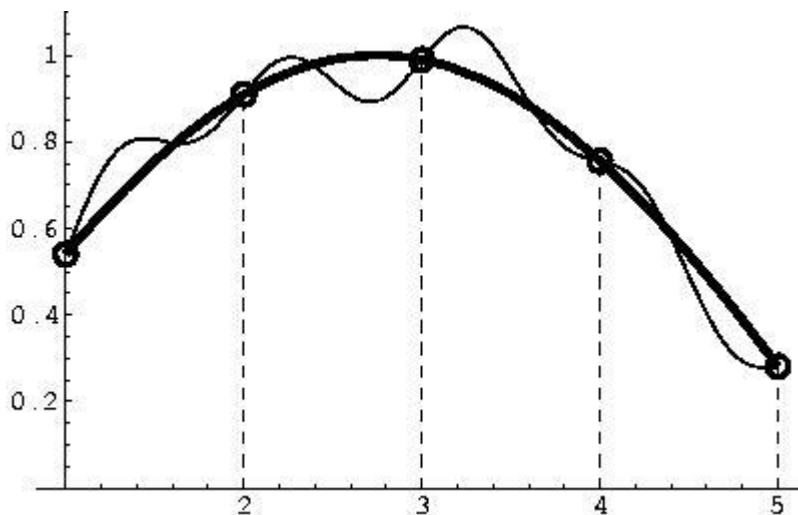


Figure 2. Digital Audio Function

The number of samples taken at each second, the *sampling rate*, should be chosen with care. The *Shannon Theorem* states that if the sampling is done twice as fast as the "fastest oscillation" of the signal then it is possible to recover the original sound; this is called the *Nyquist frequency* sampling rate.

It is important to have a good idea of the type of signal that you are processing, in order to choose an appropriate sampling rate. Large sampling rates give better quality, but also require more memory and computer power to process and store.

The quantization step sets the value of the function at each discrete sample. A simple approach would be to use a **float**, thus requiring four bytes per sample. In general this is far beyond the required quality, and people use either one byte (unsigned character) or two byte samples (signed word).

Let's compare two concrete examples to see how these choices can dramatically change the required amount of memory to keep and process the sounds. On a compact disk the audio is stereo (two different tracks, left and right), at 44,100 samples per second, using two bytes to keep each sample. One minute of it will require: $2 \times (60 \times 44,100) \times 2 = 10.5\text{MB}$ of storage.

In the case of voice mail or a telephone-based system where the controlling computer requires playback to the user, these numbers are quite different. There is no need for so many samples, since the phone lines have a serious quality limitation, usually these systems take 8000 samples per second, use only one channel (mono) and keep only one byte per sample. In the same way, for one minute we will need: $1 \times (60 \times 8000) \times 1$, about 480KB of information.

Sound Card

Since it is important to have your sound device fully functional, many configuration details should be checked, such as the sound support in the kernel (you can compile as a kernel module). Check the SOUND-HOWTO by Jeff Tranter (Jeff_Tranter@mitel.com) at <http://confused.ume.maine.edu/mdw/HOWTO/Sound-HOWTO.html>—it's a good reference.

Sound Files

Once we have the digital audio signal in hand it is important to encode it. There are many types of sound files (au, wav, aiff, mpeg), each with its pros and cons. Some of them simply stack the samples in a vector, byte after byte, while others try to compress the signal through transformations and sometimes heavy computations.

Fortunately, on sunsite (<http://sunsite.unc.edu/>) you can find **AFsp**, from Peter Kabal (kabal@tsp.ee.mcgill.ca), a library that reads and writes these files for you. I have not used it, since by the time I found it, I had already written some code to do the same operations. AFsp is very well documented.

Simple Play Operation

There are many good sources of information around the Web on how to program the input and output of sound in Linux. The first you should check out is the *Programmer's Guide to OSS* at <http://www.4front-tech.com/pguide/index.html>. It contains all the information you need for the control and manipulation of different aspects of your audio hardware, like MIDI, mixing capabilities and, of course, digital audio.

In Linux the sound hardware is generally controlled through a device (normally `/dev/audio`). You have to activate it, with a call to **open** and set some parameters (sample rate, quantization method and mono or stereo) using **ioctl** (I/O control). These basic steps for playing or recording are well illustrated in the "Basic Audio" section of the guide mentioned above.

Playing is accomplished by a **write** operation of the vector of samples on the device, while the recording is done through a **read**. There are some subtle

details that are being skipped, like the order in which these operations are to be done, for the sake of simplicity.

Streaming Audio and Interactive Applications

If you are planning to create applications that require real-time interaction (for example, a game engine) and have to continuously stream an audio sequence, there are some important measures to take to ensure that the audio buffer is neither overflowed nor underflowed.

The first case, overflows, can be solved by knowing a bit more about the OSS implementation (check the “Making Audio Complicated” page of the *OSS Programmer's Guide*). The buffer is partitioned in a number of equal pieces, and you can fill one of them while the others are in line for playing. Some `ioctl` calls will give you information about the total available space in the buffer, so that you can avoid blocking. You can also use IPC (Inter-Process Communication) techniques and create a different process responsible only for buffer manipulation.

When you send the audio to the output at a slower rate than the device plays, the buffer gets empty before you send more data causing an underflow. The resulting effect is disturbing and sometimes difficult to diagnose as an underflow problem. One possible solution is to output the audio at a slower rate, thus giving the computer more time to process the data.

Available Software

Looking at how other people write their programs helps to understand the inner difficulties of the implementation problems. So, I looked around the Net to see what was available and was quite impressed by both the quantity and the quality of the software I found.

A good, simple example of a “sound-effect server” is **sfxserver** by Terry Evans (tevens@cs.utah.edu), available on sunsite. It takes control of the audio device and receives commands (currently only from stdin) like “load a new effect”, starts playing the loaded effect, etc. In the same place you can also find **sfxclient**, an example of a program client.

Generic *network audio systems* have taken the approach of keeping the high-level, application development far away from the hardware and device manipulation. The **Network Audio System** (nas) is one implementation of this paradigm, having the same idea and framework of the X Windows system. It runs on many architectures such as Sun, SGI, HPUX and Linux. Through it you can write applications that take advantage of sound across the network, without worry about where you are actually working—the network layer takes

care of everything for you. **nas** comes with documentation and many client examples. You can download it from sunsite along with a pre-compiled **rpm** package. Some games like XBoing and xpilot already support it.

Another network transportation implementation is **netaudio** (<http://www.bitgate.com/netaudio/>). It is not intended to work as an intermediate layer between applications and devices like **nas** and is only responsible for real-time transmission of data along the Net, allowing some interesting properties like rebroadcasting. The *great* advantage I saw was its compactness: the gzipped, tar file is around 6 KB. The basic idea is to use another program in a pipe-like structure to enable playing after reception (or recording for transmission). The README file gives examples of how to compress the audio with other programs to reduce the required amount of bandwidth thus becoming a free, real-time audio alternative for the Web.

A similar package, using LPC compressing methods (designed for voice only) is **Speak Freely for Unix** from John Walker (<http://www.fourmilab.ch/>).

Another interesting example of an audio streaming application is **mpeg** audio playing. The compression achieved by this method is incredible, making high-quality audio on demand possible through the Internet. Unfortunately a fast machine is required for real-time playing.

Again, looking at your sunsite mirror, you will find some implementations. One that is fast, interesting and attention-getting is **mpg123** from Michael Hipp (Michael.Hipp@student.uni-tuebingen.de) and Oliver Fromme (oliver.fromme@heim3.tu-clausthal.de). The official web page is located at <http://www.sfs.nphil.uni-tuebingen.de/~hipp/mpg123.html>.

Conclusions

It is very important to have the basic, digital-audio concepts clear in mind during the implementation process of an application, otherwise there is the risk of a misconception or tiny mistakes that are difficult to locate.

The great number of programs available around the net show that this kind of resource is appealing, and they also provide you with a base at which to begin programming, so you do not have to start from scratch.

I have purposely left the implementation details out of this article, since it would be impossible to keep it both concise and accurate, as well as helpful. Again, I suggest to those interested in the inner details to go to the links mentioned and download some of the existing applications to learn how they were coded.

References

Siome Klein Goldenstein is an Electronic Engineer and has a MS in Computer Science (Graphics). By the time of printing he will be starting a PhD in Computer Science at University of Pennsylvania.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

The Quick Start Guide to the GIMP, Part 1

Michael J. Hammel

Issue #43, November 1997

This first of a four-part series introduces us to the GIMP, a Linux power tool for the graphics artist. Mr. Hammel used the GIMP to design this month's cover.

It has been a long time coming, but the wait is over: Linux has its first real end-user power tool. It's not for administrators. It's not for network hacks. It's not another developers tool. It's for artists. It's for media managers and graphics nuts. It's for fun. It's for real. It's the GIMP.

This article is the first of a four part series on the GIMP. This first part is a general introduction to the application including some definitions and system requirements. Future articles will cover basic functionality, an exploration of the Image Window including the use of layers and a detailed walk-through of the Toolbox.

What is the GIMP?

The GIMP is the *GNU Image Manipulation Program*, a tool for manipulating graphics images that borrows its look and feel from the popular Macintosh and Windows program from Adobe called *Photoshop*. It is used for all varieties of image processing, photo retouching and image composition. Built-in features, such as scripting, make it very easy to create logos for web pages and magazine cover art, and the Plug-In API provides a convenient mechanism for extending the very rich standard feature set. Support for a large number of input and output file formats is provided, including support for reading Photoshop files which is under development at the time of this writing.

It is nearly impossible to discuss certain features of the GIMP without first introducing others, but one must begin somewhere. I will attempt to limit confusion by mentioning features before they are discussed. Also, it will be impossible to go into complete detail of all aspects of the GIMP even in four articles. Layers, in particular, will be difficult to discuss at length without

compromising discussion of other aspects of the application. Perhaps, if there is enough interest, I can expand this series into a full sized published text. In the meantime, these articles will provide an introductory view of this wonderful new tool.

Getting the GIMP

The first thing you want is the program itself. Binary versions of the latest release packaged in an RPM format will most likely be available from Red Hat, although at the time of this writing none of the latest developers releases were available in binary. The GIMP's web site will also have binaries available for some platforms, including Linux. Of course, the GIMP is also distributed with full source code as well. The source distribution can be retrieved from `ftp://scam.xcf.berkeley.edu/pub/gimp/developers/`. There are two files needed to build and install from the source: `gimp-1.0.tar.gz` and `gimp-data-1.0.tar.gz`. The former is the real source code and the latter is the standard set of brushes, palette and so forth that comes with the default source distribution. Note that the actual version and release numbers might be different, so just look for the latest releases on the FTP site. At the time of this writing the actual release number was 0.99.10.

Building from source should be a straightforward process:

1. Unpack the archives
2. Run **`./configure`**
3. Run **`./make`**
4. Run **`./make install`**

This process is required for both the source code and the data files archives. If you intend to install in the default locations under `/usr/local`, you need to be the root user to do the last step. If you wish to install in some other location, you can instead type:

```
./configure --prefix=/
```

There are a number of options available via the configure program. Check the `INSTALL` file first, and if you can't find what you need there try running:

```
./configure --help
```

Also, as always, help is available from the GIMP Developer and GIMP user mailing lists (see the section at the end of this article for Resources on the Net).

Starting and Stopping the GIMP

Once the GIMP has been installed you're ready to start experimenting. Starting the program is simple: type **gimp**. If your system does not support the MIT-SHM extension (see below), you can type:

```
gimp --no-xshm
```

When the GIMP starts, it first reads a configuration file located in the `.gimp` directory located in the user's `$HOME` directory. This directory is where user-configurable options, user-specific plug-ins, palettes, brushes and so forth are kept. A similar set of directories are kept in the system directories (by default under `/usr/local/share` and `/usr/local/lib`).

If the GIMP does not find the `.gimp` directory in the users `$HOME` directory, it will create and populate it for them. Since this process can potentially take longer than just a few moments, a window is displayed explaining what the program is doing. The user need not do anything during this time. When the initialization of the `.gimp` directory has completed, the program opens the main window, and the user can begin work. Future invocations of the application will not include the creation of the `.gimp` directory unless that directory is removed for some reason between GIMP sessions.

The configuration file, `gimprc`, contains the specifications for the directory in which the GIMP is to find all of its files. In this way a user can, if desired, use a locally installed version of the GIMP instead of the one installed in the system directories. The `gimprc` file also defines the default settings for many of the features for the application. Such items as the speed at which the “marching ants” (the moving dashed line that denotes a selected region of an image) move, the number of colors to use, the GIMP colormap option and the Gamma correction setting are defined in the `gimprc` file. This file is well commented so you shouldn't have too much trouble understanding what the various configurable parameters affect. You should probably take a look at this file at least once before starting the GIMP, although it may not be absolutely necessary for you to become familiar with it, especially if you are using a high-end system with a high-end graphics adapter.

The GIMP uses its own temporary and swap files for saving runtime data. Most temporary files get removed at the end of a GIMP session, however some of these files may remain after exiting. It is recommended that the temporary directory be a directory that is specific to the individual user. This directory should not be one that is cleaned up by the system automatically. The swap file is a file that can get very large due to the use of the tile-based memory scheme that the GIMP uses. Because of this it is recommended that the swap file directory be `/tmp` or some other directory on a file system with plenty of disk

space. Defaults for both the temporary files directory and the swap file directory exist in the `gimprc` file. You should verify that these are appropriate for your use before doing any heavy, image-processing work.

One of the settings in the `gimprc` file that you might consider changing is the one that allows the GIMP to use its own colormap. If you have a system with 1MB of video memory (not system memory—there is a difference) and you wish to run in a high-resolution mode such as 800x600 or 1024x768, you may only be able to run with a maximum of 256 colors. In this case the GIMP (and many other applications, such as Netscape) will quickly run out of colors. In some cases, with smart applications, you'll get what's known as “color flashing”—where the color of windows and the background changes depending on which window currently holds the focus for keyboard and mouse input. Although a bit annoying, color flashing is the result of a little trick provided in X Windows applications to help guarantee they have enough colors. They do this by *installing their own colormap*. The GIMP allows you to do this in the `gimprc` file. However, this option is not turned on by default. If you have a low-end video system and run using an 8-bit (256 color) display, you'll want to uncomment this option in the `gimprc` file before starting the GIMP. If the color flashing that occurs is bothersome, there is another option for setting the color cube. Changing the values here, however, is a little trickier and should probably only be done if you understand what “dithering colors” means.

Remote Displays

As with nearly all X applications, the GIMP supports the ability to run on one machine and display on another. Most X applications accept, through the Xt toolkit, the **-display** command-line option to specify a remote host to display on. The GIMP accepts a similar command except that two dashes are required. Somewhere in the evolution of Linux there was a migration from one to two dashes for command-line options. I'm sure there is a valid reason for it—I just never figured out what it was. In any case, the command-line option for displaying on a remote system is:

```
--display <remote host>:<display number>.<screen>
```

For most users, the display number and screen will both be 0 (zero).

System Requirements

The GIMP has been ported to many of the well known Unix platforms, including Solaris, SGI, FreeBSD and, of course, Linux. The software is not hardware specific since it makes use of the low-level X11 interface provided by Xlib (GIMP's toolkit, `gtk+`, does not rely on either the Xt toolkit or Motif). This allows the X server to handle the actual screen drawing and lets the GIMP provide the

computational work of the display. Consequently, the GIMP works with all of the well known X servers for Linux (MetroLink, Xi Graphics and XFree86). As with any high-end graphics tool, you'll have better results with the high-end graphics adapters. I use a Matrox Mystique at 1152x900 with 4MB of video memory that allows up to 16 million colors. Be sure to check the server documentation or vendor's marketing material to find out if the video adapter you use is supported.

One extension to the X Window Protocols that many X servers support is the MIT Shared Memory Extension. This extension provides a method for an X application to make use of the shared memory resources of the operating system without having to go through the Xlib interprocess communication channel. For this reason, the processing of large images by the low-level X routines can be done much faster. The GIMP makes use of this feature to help provide speed enhancements. Unfortunately, not all X servers provide support for this extension. If it appears the GIMP is not behaving well, you can try starting it with the **--no-xshm** command-line option to disable the use of the MIT Shared Memory Extension. One common scenario which often needs this option is when a user starts the GIMP on one machine but has it display across the Net on another system.

Note that all three of the major X server vendors (MetroLink, Xi Graphics and the XFree86 Project) currently provide servers for Linux that support this extension. However, older versions of these servers may not. Some of the popular (but older) X terminals from NCD also do not support this extension. To find out if your server has this support, type:

```
xdpyinfo | grep MIT-SHM
```

If this command prints out "MIT-SHM", your server supports the extension. If it doesn't, you need to use the **--no-xshm** option. On the Xi Graphics AcceleratedX 3.1 server running with a Matrox Mystique video card, the shared memory extension appeared to function incorrectly and the GIMP had to be run without the use of the X shared-memory support. A small number of other cards had similar problems with this server. If you have few problems other than the windows which display images are being redrawn incorrectly (they appear to have overlapping tiles, for example), you should consider running with the X shared-memory option disabled. Note that Xi Graphics is currently working to resolve this problem, and the problem does not exist for all cards supported by their X server.

The GIMP uses a tile-based memory management system so that extremely large images can be worked on without exhausting physical memory. Even so, work like this benefits tremendously from additional system memory. I recommend you have at least 16MB of system memory to do simple web page

graphics. If you intend to get into print media you'll be working on much larger images, with X/Y dimensions of 2000x2000 or more. In these cases you need at least 64MB of memory and probably more.

CPU speed is also important for doing high-end graphics work. You can use the GIMP on any Linux-based hardware platform, but you may get frustrated waiting for some pixel operations to complete on older systems. For example, operations like blurring, mosaics or adding sparkles to regions of images can be very CPU intensive. I used a 486/66DX2 (i.e., a speed doubled 486-33MHz) system with the original release of the GIMP well over a year ago. The new version benefits a great deal from the added power of the Cyrix P166 (133MHz) system I currently use. The equation for computer graphics is simple: faster CPU + more memory = reduced frustration.

The GIMP does not at the time of this writing have direct support for scanners, tablets or pen devices. Work is currently underway on a generic scanner interface (called SANE) which includes a plug-in for use with the GIMP. This plug-in is not part of the base distribution at this time. I expect that scanner support will be much more extensive for the GIMP by the time this article reaches the newsstand.

Support for Wacom tablets is also in the works. A number of contributors have the Wacom ArtZ working with XFree86, and a couple of individual projects were started to integrate the tablets with the GIMP using the X Input Extension. Recently (in June 1997), these projects began to merge. Again, by the time this article reaches the newsstand, this support should be much more extensive. It is interesting to note that Wacom is a commercial supporter of the XFree86 project. At the time of this writing based on information from their web sites, neither the MetroLink nor Xi Graphics X servers appear to support Wacom tablets.

Disk Space Requirements

The version of the GIMP that I use, the developers 0.99.10 release, requires approximately 19MB of disk space for the runtime files. This includes about 11MB for the binary, 5MB for the various plug-ins and about 2.5MB for the libraries and data files, such as patterns, brushes and so forth. The source code takes up approximately 11MB uncompiled.

Working with images is very disk space intensive—this means that you need to have a lot of disk space for all the images you will be creating or manipulating. You also need space for variations of images, photos from archives to use as starting points for creating your images and a number of copies of the same image in different formats. Managing all the files alone could be a full-time job

(and in large media shops, it is a full-time job), so be aware that in the long run you may need to add hard disk space.

Moving On

Now that you know a little about what the GIMP is, where to get it and what type of hardware and software requirements and support it has, you're ready to look at the application itself. Next month we'll discuss some basic features such as file I/O, dialog windows and cursors. Each of these topics could fill a chapter in a book, but we're going to try to cover them in one article. So hang on, we're just getting started.

References



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

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

GIF Images on the Fly

Jimmy Ball

Issue #43, November 1997

This article gives us a quick lesson in creating new images from old.

One of the best kept secrets on the World Wide Web is how GIF images are created in real time or “on the fly”. Some of the reasons for creating GIF images on the fly include highlighting a map image, combining several images into one (similar to the layers feature in CAD packages) and even making virtual postcards.

My reason for developing programs to create GIF images on the fly was for a campus map that would highlight important locations. That is, when anyone visited our campus map web page and selected a certain building or site to locate, we wanted to highlight that particular site in some way.

As with most projects, all of the details had not been worked out in advance, so I chose a development route that would save me time—I decided to write a program to automate the task. Programming is more fun for me than editing 100 or more images, especially when people tend to change their minds. It turns out that the development time was not dramatic, and when modifications were requested, it was a trivial task.

In this article, I simplify my project and outline the steps to create a new GIF image by adding an arrow to an existing image. You need to be familiar with CGI programming in Perl as well as HTML. You do *not* have to be an expert.

Getting Started

First, download and install the items listed below.

1. **Perl:** This is the programming language we will use. For CGI programs, I typically start out with a Perl program and then migrate to C if the speed of the program becomes the bottleneck. You will need at least version

5.002 of Perl. Download it from <http://www.perl.com/perl/>, if you do not already have it installed. There should also be pointers to the Windows 95/NT version of Perl at the URL above. As far as I know, all Perl implementations are free.

2. **GD.pm interface:** This interface provides the graphics commands to do the GIF image manipulation. It was developed by Lincoln Stein and is based on the gd C library developed by Thomas Boutell (<http://www.boutell.com/gd/>). You can get GD.pm at <http://www-genome.wi.mit.edu/ftp/pub/software/WWW/GD.html> for free.
3. **Web server:** This example was developed on the Apache 1.2 web server. However, as long as the web server allows you to write CGI scripts in Perl, and Perl is installed as a CGI language, it should work. You can get Apache at <http://www.apache.org/> for free. There are other freeware and commercial web servers, but Apache gets my vote for best web server on a Unix platform.
4. **cgi-lib.pl library (optional):** This is a set of Perl routines for reading input from a form. If you plan on accepting input from a web form to generate the GIF image, you need this or a similar package. cgi-lib.pl can be downloaded from <http://www.bio.cam.ac.uk/cgi-lib/> for free. Similar packages are available, but cgi-lib.pl is my personal preference.

Example #1

Once you have all the packages installed, you are ready to create your first CGI script that uses GD.pm.

In the first example, we read in two GIF images: a main image and an arrow image. Based on input from the user, we “place” the arrow on the main image. Last, we display the final image. [Listing 1](#) shows the source code for this example.

Line 2 requests the use of the GD interface and Line 3 verifies that we are using Perl 5.002 or higher. Line 5 includes the routines from cgi-lib.pl for reading form input. One of those routines, **ReadParse**, is used in Line 6 to get the form input data. The data from this form is stored in the associative array named FORM.

Next, Lines 8-11 set up variables which include the filenames for the main and arrow images. In addition, the variables for placement of the arrow on the main image are assigned default values, if they are not provided from a web form or other technique. I will show an example of passing variables to a script later.

Lines 13-15 and 17-19 do the same thing. The first set reads the main image into a variable called **\$main**. The second set reads the arrow image and stores

the image in the variable **\$arrow**. In each set, you will notice a routine **newFromGif GD::Image(GIF)**. This is the GD function that reads an image from the file and stores it into a variable. The nice thing about this routine is that it does not require image dimensions, although you probably want to know them.

Line 21 does most of the work in the script. It takes the arrow image and stores it at the x-pixel position `$FORM{xp}` and the y-pixel position `$FORM{yp}` on the main image. The last four parameters for the function define how much of the arrow image is copied. In our case, all of the arrow image was copied onto the main image.

Finally, Lines 23 and 24 output the image as if it were being called from an HTML file and Line 25 ends the script.

Now, we need to call the script to generate the image. You can do this in a couple of ways. First, just call the CGI script directly from a web browser like Netscape. An example URL would be `http://bodock.vislab.olemiss.edu/cgi-bin/example1.cgi`.

Calling the script would simply display the image on the screen. You can also put this URL in an IMG tag of an HTML file like the following:

```
<IMG SRC=
"http://bodock.vislab.olemiss.edu/cgi-bin/example1.cgi"
ALT="">
```

Last, you can modify the URL slightly to specify an x,y position for the arrow image. An example URL within an IMG tag would be as follows:

```
<IMG SRC="/cgi-bin/example1.cgi?xp=10&yp=40"
alt="">
```

Notice I took off the full URL. Doing this assumes that the CGI script is on the same machine as the web page.

Example #2

The next example uses a few more of the functions supplied with the GD interface. We draw the arrow rather than read it from a file. By drawing the arrow, we can control the angle of the arrow which is drawn on the main image. [Listing 2](#) shows the source code for this example.

Lines 1-8 should look familiar from the previous example. We simply define variables for the main GIF image and the position/angle of the arrow. Just for simplicity, I did not include the **ReadParse** routine, so no user input is allowed.

As in the first example, we read the main image into the variable **\$im** in Lines 10-12.

Line 15 is new. This line simply allocates a color based on RGB values which range from 0 to 255. The color blue has a RGB value of (0,0,255). We can then use the color later when drawing the arrow.

Lines 17 calls the function for creating the arrow. This routine starts on Line 26. In this routine we create a new polygon (an arrow) and then spend most of the time finding the points for the polygon. After we find an x,y position for a corner of the polygon, we put it in the polygon object, **\$poly**.

As mentioned earlier, our polygon is an arrow. Line 37 creates the polygon variable. Then Lines 38-44 find points for the polygon. Here is how that section works. Lines 33-34 define the x,y positions for an arrow that points north. Since we may want to rotate this arrow, Lines 38-44 come into play. These lines figure out the new x,y positions of each point taking the rotation angle into consideration. Line 43 adds the point to the polygon taking into consideration the size of the image (subtracting 75) and the x,y position where the arrow will be placed (**\$xpos** and **\$ypos**). Rather than spending all day trying to explain this section of the code, take a look at an old CGA graphics book. In fact, I got the formulas for this section from a CGA graphics manual.

At last, we have a polygon stored as the variable **\$poly** which is returned to the main program. Line 19 is another GD interface call to fill the arrow with our color and place it on the main image. Finally, the image is displayed and the script exits.

Like the first example, you can call this CGI script both directly and via an IMG tag as shown here:

```
<IMG SRC=
"http://bodock.vislab.olemiss.edu/cgi-bin/example2.cgi"
ALT="">
```

If you want to dynamically control the angle of the arrow, you would need to modify this example to include the `cgi-lib.pl` library, read in the data with **ReadParse** and then define **\$angle** appropriately.

A Little More Detail

We now know how to create a GIF image on the fly and view it both directly and from an IMG tag within an HTML file. What if you need to save the image?

Of course, first of all you need a directory where the images will be created. Depending on your web-server configuration, the ownership and write

permissions of that directory will need to be modified so that the CGI program can save the image. Discuss this matter with your webmaster.

Once that issue is resolved, your CGI script can output the image to a file using a unique, file-naming scheme. To recall this file name, you can use a database or simply inform the client of the name of the file. For example, if you develop a CGI program to create virtual postcards, the only person who needs the name of the image is the sender and/or the recipient of the postcard. This could be displayed within the web browser once the postcard has been created or e-mailed to the recipient. In other situations, several people may need to view the image. In this case, a database for storing the file name with the associated options is the best choice in my opinion.

In either case, storing the image can be done in addition to displaying it. That is, just use regular Perl I/O statements to print the image to a file. Pick up any Perl programming book and check out the File I/O section.

Last, you might have to keep a watch on the directory where the images are stored. If the file names are unique and the images are large, consider deleting old files periodically to avoid buying a new disk each month. Then again, you might want to use that as an excuse to obtain more disk space.

For Further Information

Of course, the documentation that comes with GD.pm can fill in any gaps that you might have. In addition, here are two URLs that show some of my programming work in creating GIF images on the fly. The first is a campus map for the University of Mississippi (UM). The second example highlights a few interesting sites about my home state.

For the UM campus map, the original GIF creation programs are written in C and use the gd C library. You can visit that page at the URL <http://www.olemiss.edu/hospitality/maps/>. The **Find** option at the top of the page will take you to other web pages. From there, you can select a site of interest. With a simple mouse click, an image is displayed of the campus with an arrow to highlight your selection. In addition, all the information on each building is listed in a database, so the CGI program has to first get this information before the image is created.

A more elaborate example of GIF images created on the fly can be found at the URL <http://bodock.vislab.olemiss.edu/onthefly.html>. This combines several options such as text, a choice of colors, a variety of pointers, etc. Be sure to follow the "More Practical Example" link at the bottom of the left-most frame. That section shows examples of adding several text and highlighters at the same time. In addition, the map is clickable thanks to some HTML code for a

client-side image map. The CGI programs for this application were developed with Perl and the other packages mentioned earlier.

In Closing

As with any development project, everyone has their opinion of the best package to use. I have been very satisfied with the gd C library and the GD.pm interface for creating GIF images on the fly. You may find other packages or interfaces that suit your needs better depending on your experience and your web server platform.

My only advice comes originally from my Dad, but slightly modified for the nineties and GIF image creation: "Never use a wrench (image editor) when an air ratchet (programming with GD.pm) can do the job."

Jimmy Ball is a Supercomputer User Consultant with the Mississippi Center for Supercomputing Research located on the campus of the University of Mississippi. He also freelances as an Internet consultant focusing on web development, Unix administration and training. Jimmy can be reached by e-mail at ccjimmy@mcsr.olemiss.edu or jball@ebicom.net.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Linux Expo at Union Bank of Switzerland

Martin Sjoelin

Issue #43, November 1997

Promoting Linux in the workplace...

During the summer of 1997, employees of the Union Bank of Switzerland (UBS) put on a one-month expo to showcase Linux. This article describes how and why we did it.

The Union Bank of Switzerland (UBS) is one of Switzerland's three largest banks, with offices world-wide and 30,000 employees. Currently, UBS has 25,000 PCs running mainly Windows 3.11, 800-900 Novell NetWare servers (3.11 or 4.11), around 800 applications servers, around 1,000 workstations running Solaris and at least 3,000 printers.

I work in the system development section (SYBS—System and Basic Services) where we mainly use workstations and servers running Solaris 2.5.1. We produce our own “repackaged” Solaris release, named OpenLAN, which is used on the production Unix servers. The OpenLAN release includes Perl, tcl/tk, et al. —the normal set of Unix tools in a package format. Notice the word “package” above. All software is delivered in a Solaris (Sys V) package for easy installation and removal.

Figure 1

Why Linux?

Quite a few of the employees have a PC at home and for those working with or on the Unix side, it is (or would be) quite natural to install Linux at home. Linux provides an ideal environment for learning more about Unix in general, system administration and Unix tools. Also, a Linux system is neither resource hungry nor expensive to set up. Today it is possible to buy a shrink-wrapped installation on a CD-ROM, and you do not need to be technical expert to install Linux on your machine.

Preface

In the building where our section is located, there is an area which is used for one-month expositions about different subjects. The subjects range from information about different parts of UBS, security policies or employees' personal interests: astronomy, model plane flying, etc. The available area is approximately four by four meters with two panels for posters.

Last summer, Alois Camenzind thought that a small Linux expo would be a good idea. This would enable us to spread information about Linux and its uses—let others know that Linux is a “real” Unix system. After getting the “Okay”, Camenzind contacted employees within the section who had expressed both knowledge and interest in Linux. We formed an unofficial workgroup to prepare the Expo. The main work was producing posters with information about Linux and Linus Torvalds, getting resources (machines), setting up the machines and contacting interested companies for information and products.

Figure 2

The Linux Expo

We ended up with two Intel 486 Compaq boxes which were set up running Linux kernel version 2.0.12 (the latest stable release at that time). We used the Red Hat Rembrandt Beta II distribution provided to us by Red Hat. A SPARCstation 5, running kernel version 2.0.16 with the SPARC Red Hat distribution, was also on display to show multiple platform support.

We set up an Apache 1.1.1 WWW server running on one of the Compaq machines. This WWW server had a preloaded cache with HTML pages from the SSC web site, <http://sunsite.unc.edu/>, the Swiss Linux User Group and other interesting Linux-related web sites. Since we were behind a firewall, we opted to load the cache on an Apache proxy server to make information generally available during the Expo.

Figure 3

The same machine also served as an NFS server for the smaller Compaq on which we stored the binaries and sources. The NFS server made it possible for interested people within UBS to obtain more information via HTTP or to mount the exported file system via NFS to get up-to-date kernel sources.

X11 was running on all machines, as well as a subset of the application packages provided in the distribution as an example of what kinds of software are included in a typical Linux distribution. To show an example of a commercial application running under Linux, we installed a demonstration

copy of Applixware. We configured the machine to start Netscape whenever a user logged in, displaying our WWW server home page with pointers to the cached pages and other collected material. This was our invitation to the user to start browsing and discover more information about Linux.

Figure 4

Behind the machines (see photographs) were the panels with posters about Linux (all in German): "What is Linux?", "Information sources for Linux", "The GNU General Public License (GPL)", "Linux Development History", "Linux Time Line" and "Who is Linus Torvalds". We also made information provided by SSC and Red Hat available beside the machines.

During display, anyone could just walk up to one of them and log in. Then (s)he could playing with the system, start X applications or just browse the local web via Netscape. We also provided a mirror of the latest release of the kernel as well as a Linux distribution which the user could copy.

Summary

The machines and materials were on display without anyone from the workgroup watching or being there, other than during lunch time or coffee breaks. The whole idea was to have a display which invited people to sit down and explore the Linux systems.

Walking by the area at different times during the day (or for some of us—nights or evenings), there were always people playing with the Linux machines. Taking into account that most of the material (SCC flyers, etc.) put on display had to be renewed every week, we felt people were quite interested in what Linux had to offer.

Most of our time was spent getting the information about Linux and its history from the Net, preparing the demonstration machines, making the presentation panels and setting up the WWW cache. Once everything was in place, we only had to update the mirror material and occasionally reboot the machines to insure proper operation.

Acknowledgements

Martin Sjolín moved to Zurich (Switzerland) in July 1995, where he works at UBS dealing with System Management issues for Solaris machines. He has recently begun to explore NT 4.0. In his spare time he enjoys telemark skiing, cooking, running, windsurfing, reading and, of course, hacking his Linux systems. He can be reached via e-mail at marsj@sjolin.ch.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Sound Through the PC-Speaker

Paul Dunne

Issue #43, November 1997

Here's how to get sound to your speakers using a driver instead of a sound card.

Linux supports most of the popular sound cards. If you don't have a sound card, you can still get a degree of sound support from the humble speaker that came with your PC. In this article, I will discuss one way of obtaining sound output without a sound card.

What PC-Speaker Is

PC-Speaker is a driver for the modest sound output device that comes standard with most (perhaps all?) IBM PC clones. It is installed as part of the kernel or as a loadable module; either way, the kernel needs to be changed. PC-Speaker comes with a small set of programs to use with it—I have compiled these programs on my system without trouble.

Modifying the Kernel

The driver comes as a patch file, which must be applied to the Linux source directory (`/usr/src/linux`). When `make<ls>config` is run after applying the patch, it will ask whether you want PC-Speaker support—answer “yes”. Give the commands:

```
make depend; make clean; make zImage
```

and your new kernel will be ready. The patches to the source include some header files for `/usr/include/sys`, which are necessary to make the utilities that accompany the driver.

Devices

The driver supports the following devices :

1. /dev/pcsp: The raw data device
2. /dev/pcaudio: The SUN-audio device
3. /dev/pcmixer: The mixer-device

I have /dev/pcsp only defined on my machine:

```
crw--w--w-   1 root  root   13,   3 Aug 27 20:25 /dev/pcsp
```

pcsel

The program **pcsel** sets options for PC-Speaker and is used to configure the /dev/pcsp device at system startup and to test new devices. You can also assign an output device to /dev/pcsp using the **pcsel** program. The supported output devices are:

- Stereo-on-One (designed by Mark J. Cox), which is auto-detected during kernel startup and selected by default.
- PC-Speaker, which is selected if Stereo-on-One was not found.
- Mono DAC, which is for one Ip-port.
- Stereo DAC, which is for two Ip-ports.

Listing 1. Help Output from ptsel

Specifying the help option on the ptsel command line:

```
$ ptsel -help
```

gives you a listing of all the ptsel options and what they mean. With no options specified, ptsel reports the actual output-device and its parameters in this way:

```
$ ptsel
PCSP driver version 1.0
Actual PCSP output device: PC-Speaker
Volume : 100 %, real samplerate : 18356 Hz
Maximum Samplerate is 51877 Hz
16bit Stereo Emulation enabled
```

vplay and vrec

These two programs, **vplay** and **vrec**, can be used for recording and playing the following types of files:

1. Creative Labs voice files
2. Microsoft wave files

3. raw audio data files

Both programs accept the same options, which can be listed by specifying the help option:

```
vplay --help
```

The output of this command is shown in [Listing 2](#).

Applications

Okay, confession time—the main reason I had for adding this driver to my kernel was to have sound effects in Doom. Here is another trivial example of what you can do with PC-Speaker. I have a directory of .wav and .au files. This shell script, called from my .profile file, plays one of these audio files at random each time I log in.

```
#!/bin/sh
# random-sound.sh: play a random file from the
# sounds directory
export count=`ls sounds/*|wc -l|sed s/ //`
export count=`expr $count + 0`
(1>/dev/null 2>&1 vplay `echo sounds/*| \
awk BEGIN{srand()}{x=1+int(rand()*number)
print $x} number=$count`) &
```

Where to Get It

The latest version of PC-Speaker can be found at <ftp://ftp.informatik.hu-berlin.de/>, in the directory `/pub/os/linux/hu-sound/pcsnd*` (where `*` represents the latest version number).

Conclusion

PC-Speaker is a neat bit of software that makes good use of the basic speaker. Be warned—you will get plenty of interference if your computer is *digitally noisy*. It is unlikely to become part of the standard kernel; the author, Michael Beck (beck@dgroup.de), tells me that one of the reasons is that the driver interferes with the Linux clock. I haven't noticed this myself, but my clock is re-synched with my ISP's clock four times a day. The distribution comes with a file that describes how to build your own sound output devices. For example, using these instructions you can connect your PC to your amplifier.

Paul Dunne is a free-lance writer and consultant who specializes in Linux. His first book, *Linux for Webmasters*, is due to be published by Digital Press later this year.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

YODL or Yet Oneother Document Language

Karel Kubat

Issue #43, November 1997

Would you like to publish the texts you write in more than one format—PostScript for printing, plain ASCII for e-mail, and HTML for the a web page? YODL could be just the language you need.

I wrote the first version of YODL in late 1995 and early 1996, as a general document language for my personal use then named DOM, for “document maintainer”. I was dissatisfied with the SGML converter I was using at the time, and wanted to write a quick-and-dirty program that would read a document and convert it either to LaTeX (for printing PostScript) or to HTML.

Unfortunately, a quick-and-dirty program, like a boomerang, has a way of flying back at you. As soon as I began to use YODL, I told my colleagues at work about it, and they were immediately ready to use it too. Soon I was rewriting YODL, implementing new features, writing converters for other formats and documenting it all. YODL has evolved from a private program to the document language everyone in my department uses. I consider YODL ready for the world in its current “beta” state. In this article I'd like to introduce YODL and tell you why it is a handy program to have around.

[Obtaining and Installing YODL](#)

Design Specifications of YODL

First, let me explain my goals in designing YODL. Most importantly, I wanted YODL to become a document language that would be intuitive and easy to use, unlike many common document languages such as LaTeX, HTML and some current SGML implementations.

In each of these languages, certain characters are special. In LaTeX, for example, you can't simply enter `$%*` and expect it to appear in your text. In HTML you must enter `<` as `<` to produce the `<` character. In my version of SGML

you must enter an ampersand (&) as either **&** or as **&ero;**, depending on whether you want the ampersand to appear in normal text or in a listing—which is horrible. On the other hand, all characters in YODL appear in the output as you entered. My idea is this: just type away, and whatever you type goes into the output. YODL implements translations such as these with *character conversion tables*. For example, the LaTeX conversion table specifies that a ***** in the input results in a ***\$** in the output. (You could even create a character conversion table stating that aa should lead to bb, bb to cc, and so on; which might produce interesting output.)

“Whatever you type goes to the output” is, of course, relative. Each document language must provide a way to put commands in the text to do things such as change the font, start a new chapter, etc. Typing commands in LaTeX, HTML or SGML is, in my opinion, awkward. For example, typesetting text in boldface requires that you enter **{\bf text}**, **text** or **<bf>text</bf>**. I don't know about you, but my fingers always get stuck when typing the curly braces, backslashes and smaller-than or greater-than characters. If you program on a regular basis, you will probably agree that typing parentheses is easier. Therefore, I chose to define a command in the YODL language as a macro name, followed by arguments in parentheses, as in **bf(text)**. This syntax looks like a C-style function with an argument list, except that macros having more than one argument will have each argument within separate parentheses. Another advantage of parentheses is that many editors have a “match-mode” that highlights pairs of these characters making the typing of text much easier.

As for the “usefulness” of a document language, however arbitrarily measured, I find that such a language must support at least automatic numbering of sections, labels in the text and references to them, and it must create a table of contents. Respect where respect is due: these features (and lots more) are implemented in LaTeX. I, therefore, chose to let YODL “emulate” this feature in other output formats. When YODL converts a document to HTML, it creates a clickable table of contents and numbers its sections.

By design, the YODL package consists of one generic program that is able to process simple commands—this is the **yodl** program. This program is not a real converter by a long shot but just a first phase. The bare **yodl** program knows about commands such as **INCLUDEFILE** (to read in a file), **DEFINEMACRO** (to define a new macro), **IFDEF** (for conditional execution), etc. A real converter uses **yodl** for its first phase but supplies macro files and character conversion tables for a given format. For example, **yodl2tex** which converts a YODL document to LaTeX, loads an appropriate macro file specifying that **bf(text)** is in LaTeX **{\bf text}** and that a **\$** character must be typeset as **\\dollar\$**. Some converters, like the HTML converter, require a post-processor for specific

actions, e.g., to resolve labels and references and to create a clickable table of contents. Normally, the user is not aware of such peculiarities: simple shell scripts (such as, **yodl2html**, **yodl2tex**, **yodl2txt**) run the **yodl** program, supply the right macro files and, if necessary, start post processors.

The last design consideration I want to mention is that situations can arise in which you **must** use commands in a given output language (LaTeX, HTML or whatever) to accomplish special goals. Although YODL can be used without knowledge of the output format, the final document language is by no means hidden. The macro package implements the commands **latexcommand(cmd)**, **htmlcommand(cmd)** etc., which are hidden. The macro package implements only the commands active for their output format. This means that all the nifty features of YODL can be used for the “standard” things, while you can always fall back on the specific commands of the final output language for special features.

Using YODL

The macro package that comes with YODL defines four main types of documents: articles, reports, books and man pages. A YODL source file must always mention what type of document is being processed. The difference between the document types is that different top-sectioning commands are defined for each (e.g., an article has no chapters, a report has no parts and a man page has its own specific sections) and how the HTML converter splits the resulting HTML file into different sub-files. The HTML output is split by chapters, which are reached via a clickable table of contents and by “next chapter” links. The statement that defines the document type also sets the document title, the author and the date. (If you're familiar with LaTeX then all this probably rings a huge bell. Yes, I'm guilty of “borrowing” wherever I can.)

Before stating the document type, several optional commands can be specified to alter the appearance of the document. Examples are **mailto**, a macro that sets the default e-mail address for HTML output, or **htmlbodyopt**, a macro that is used to define the foreground or background colors. A sample document could therefore be started as:

```
htmlbodyopt(fgcolor)(#0000B7)
mailto(karel@icce.rug.nl)
report(Xwatch: Watcher of logfiles
      under an X session)
      (Karel Kubat)
      (1996)
```

These statements would be followed by the document. Reports are divided into chapters, chapters into sections, sections into subsections, etc. When labels are used following the sectioning commands, references can be made to those labels. In HTML the references are clickable links:

```
chapter(Introduction)
This is the introductory chapter. Specific
information about the installation is in
chapter ref(install).
chapter(Installation) label(install)
This chapter describes the installation in
detail.
```

As an example of user-defined macros, consider the following. Let's say that you want to typeset $1/4$ either as shown here, or as a fraction with the 1 above the 4. The layout of this fraction will depend on the output format, with only LaTeX supporting the latter notation. The way to accomplish this would be to define a new macro, say **quart**:

```
DEFINEMACRO(quart)(0)(\
  latexcommand(\frac{1}{4})\
  txtcommand(1/4)\
  htmlcommand(1/4)\
  mancommand(1/4))
```

This statement defines a macro called **quart** having zero arguments. Macros can have up to thirty-five arguments. The macro expands to a command in each output format, LaTeX, HTML, man or plain ASCII—one per conversion. Now, the new command can be used as:

```
a quart is quart() gallon
```

Also, note that the backslash character can be used to split long lines into readable source code, as in shell script programming. It's just one more feature that I considered handy and, therefore, implemented.

The YODL language has tons of other useful macros—there's not space for a complete enumeration. However, here's a couple just for the fun of it. You can use footnotes in a document by specifying:

```
footnote(
```

Depending on the output format, the text is either a real footnote or it is placed in the running text. Similarly, the macro:

```
url(
```

puts a clickable URL in a HTML document or mentions the description and location in other types of output. And there's lots more, including the way YODL handles accent characters. All can be found by reading the documentation that comes with the YODL archive.

YODL's Macro Files

What YODL Still Needs

As I said before, I now consider YODL to be in a beta state, and I've released it to whoever wants it. One of the great things about the Linux community is that

it is a pool of millions of beta-testers. I expect YODL to continue to evolve over the next few months.

In any case, what YODL still needs is a flawless converter to plain ASCII via the **groff** route. I wrote two converters: one that interfaces to the **man** format and one that interfaces to the **ms** format. That's at least a good start; though the **man** and **ms** packages can do more than just what I've picked out. Furthermore, the HTML converter only supports "standard" HTML, no nifty features such as frames, although there is some rudimentary table support. And, of course, converters to other formats would be welcome, like a converter to the **texinfo** format for the creation of info files, etc.

All in all, YODL has already proven itself in our department, where almost all documents that accompany our software are now written in the YODL language. I hope you won't find YODL worthwhile looking at, or I'll be swamped emptying my mailbox for the next few months and improving the YODL package. All kidding aside, I hope this article has tickled your curiosity enough, that you'll try out YODL and discover its usefulness.



Karel Kubat lives in the northern part of the Netherlands with his family, cats and 1967 Volvo Amazon. He's a full-time Linux fanatic, who can be reached via e-mail at karel@icce.rug.nl.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

***Linux Journal* Interviews Robert Nation**

Larry Ayers

Issue #43, November 1997

Larry Ayers, author of many articles for both *Linux Journal* and *Linux Gazette*, interviews Robert Nation, the creator of FVWM and RXVT.



[The Interview](#)

Robert Nation's name crops up often in the man pages and other documentation of Linux software. He was involved in the early development of several key pieces of Linux software such as FVWM—software that is still being revised, updated and run by thousands of Linux users. For this reason, I approached Mr. Nation to find out what he thinks of the changes made to his software and to Linux in recent years.

Larry: Although I've seen your name listed in various pieces of software documentation, I realize that I know few personal details. Would you fill me in on your background?

Robert: I grew up in Ithaca, NY, where my father is a professor at Cornell University. I received my bachelor's and master's degrees in Electrical Engineering from Cornell. Since then I've been worked at Raytheon, in Massachusetts, and Sanders, in southern New Hampshire. I've been living in Merrimack, NH for several years now, with my wife (also from Cornell and upstate NY) and three children—ages 4, 5 and 7.

Larry: What sort of work are you currently pursuing?

Robert: I work for Sanders, a Lockheed-Martin Company. Sanders is a leading supplier of electronic counter-measures equipment for the U.S. military. Sanders also works in communications and battle-field situational awareness areas. I've been working on systems to detect and classify helicopters and tanks at long range from the sounds they make and to identify scuba divers from their echo in a high frequency sonar.

Larry: How did you get involved with programming free software?

Robert: After I started work at Sanders, I realized that Unix workstations were an excellent tool for signal analysis—in field tests, we record data from some prototype radar, sonar or acoustic system, then load the data onto our workstations to figure out the best approach for performing computer-aided detection and classification of the targets.

Around 1990, I saw Linux mentioned on the Net and thought that maybe a low-cost PC could be used for the signal analysis—with the advantage that PC Laptops are cheap. They can also be easily taken on field tests for on-site data analysis to tune systems parameters while we are actually in the process of testing the system. I bought a PC and started learning about Linux to track this trend—and also brush up on my computer and software skills at the same time.

Larry: How did you become motivated to create FVWM, now one of the most widely-used window-managers in the Linux community?

Robert: My original reason for creating FVWM was to help me view spectrograms of recorded data. A spectrogram is a plot that shows time on the horizontal axis, and frequency on the vertical axis. The brightness of each pixel in the plot indicates the amount of power that was received at that frequency and time. I like to complicate matters by using color to indicate the direction of arrival of the data too. Anyway, I started looking at bigger and bigger spectrograms—until they were eventually 2000 pixels by 400 pixels, and I didn't want to lose resolution by squeezing the image onto one screen. TVTWM or VTWM would have been good enough for the job, but I wanted to learn something about X11 programming, so I started ripping apart TWM to create FVWM.

Larry: What do the acronyms FVWM and RXVT stand for?

Robert: FVWM stands for Feeble Virtual Window Manager. The original releases (around version 0.8 or so) had almost no user selectable features, so it really was feeble. I think the only thing that you could change was the color of the window borders.

RXVT is Rob's version of XVT. I believe XVT stands for X Virtual Terminal.

Larry: How do you think FVWM compares with TWM as far as size, speed, and memory?

Robert: Although a few early versions of FVWM were smaller (less memory usage) than TWM, I'm sure that this is no longer the case. Also, people tend to use FVWM modules, which consume even more memory.

TWM was not too much of a memory pig in the end—except that it initialized a table of possible mouse-button and shift/alt combinations that was fairly large. That table was used to store binds of menus to mouse buttons. I believe that TWM even initialized the table, so it really had to be allocated and then largely swapped out. After I discovered that problem, FVWM didn't have much advantage over TWM.

As far as speed is concerned, TWM uses less decorations, so it is somewhat faster. This doesn't seem to make much difference with today's fast displays, though.

Larry: What do you think of the newer FVWM variants which have been proliferating recently?

Robert: I use FVWM-95 now. I think that the task-bar is a good feature, and motif appearance/compatibility is a bit outdated in this new Microsoft era. I haven't spent a lot of time looking at Afterstep or the other variants of FVWM, but I did notice an increasing trend to use shaped color pixmaps for a lot of gadgets, and these really slow down the X terminal that I use at work.

Larry: What were the progenitors of RXVT, and what compelled you to begin developing it?

Robert: RXVT is based on XVT, which is still available on the net. XVT worked fine, but was a little slow at updating the screen. I poked around for a while, mostly to learn more X11 stuff. I also added the option for in-line graphics, which I thought would be really useful for scientific plotting—you could issue a plot command, see the results and have the plot scroll away like any other text. Although I've used that feature quite a bit, it didn't turn out to be as useful as I had hoped. I do occasionally use **Xgraph** to look at a bunch of scatter plots and end up with a million plots all over the screen. The graphics capability in RXVT does eliminate that problem.

Larry: Browsing through a book of Linux manual pages, I saw your name in the credits of the **top** program. What was your involvement?

Robert: There was a version of top that directly read kernel structures to get the process information, but it needed to be recompiled every time the kernel changed and had to be updated frequently. There was also a brand new version of the /proc-based **ps** program. All I did was combine the two, taking a lot of the display software from the old version of top, and the process information software from ps, to make the /proc-based version of top that you see now. It took about two evenings, and other people have maintained it since I first distributed it.

Larry: Have you worked on any other Linux software other than the programs mentioned above?

Robert: Nothing that I have distributed. A few months ago I tinkered with a small graphics server, similar to **mgr**, but with the ability to update windows other than the top-most window and to update several windows at once—not just the active window. I got the windowing system and text windows working in monochrome, but never finished the rest of the graphics capability.

Larry: Do you have plans to finish the graphics server?

Robert: Probably not, since I just bought a new home computer with a whopping 16MB of memory, and X runs very nicely on that equipment. It was an interesting learning experience, though. The software needed to figure out where windows overlap and update them correctly and efficiently is not trivial.

Larry: What were your first experiences with and impressions of Linux?

Robert: I started using Linux in about 1990. I think it was version 0.9 or so. I was very impressed—I installed an SLS distribution which I downloaded to the Sun computers at work and installed from floppy. As I recalled, it worked perfectly. I still use that same installation at home, upgraded piecemeal to the latest kernel, compilers and libraries. Throughout all those years, I only remember one kernel that had significant problems and had very few problems installing packages.

We've just started using Linux at work now. There is a significant issue in system maintenance—since the Suns come with complete but rare updates, and Linux comes with continuous, piecemeal updates, we don't know if it's cheaper to pay the extra money for Sun hardware and save the cost in system administration, or just buy PCs and take the system administration risk. We're using it experimentally now and will be taking a prototype submarine communications system out to sea next year, using a Linux laptop as a console for a rack of VME equipment.

Larry: Why was Linux chosen for this communications system?

Robert: We're using Linux to run the graphical displays (X11). Sanders has been using X11 and Sun workstations for displays for a long time now, so we have a significant knowledge and experience base. Our choices for the submarine system were:

1. Connect a CRT monitor directly to the Real-time system—this is acceptable, but its hard to get bulky CRTs into a submarine—they have to load through a 30 inch diameter torpedo loading hatch, which is about 2 stories high.
2. Use a separate Sun workstation—same problem
3. Use a Sun laptop—very expensive and a little slow.
4. Use Windows—we have very little programming experience.
5. Use Linux—we made this choice because of low cost, good level of in-house programming experience and software source code that is generally compatible with Sun software.

Larry: Any thoughts about Linux or other subjects you'd like to add?

Robert: I don't know how much more you could ask for in Linux—the developers have done a fantastic job of getting this operating system together. I'm glad to see that some software vendors are supporting **matlab** versions of their software now, but I wish there was more available. I wonder if there is a market for someone to license limited rights to software like Word or PowerPoint, and then port, sell and support it for Linux.

I would like to know if anyone is working on a next-generation user interface. I suppose that this could be voice activated, or more interesting, some kind of 3-D, maybe virtual reality thing. I keep thinking of the brief excerpt from Jurassic Park where they showed an interesting 3-D file manager and wondering if there are any ideas there.

Larry: Thanks for your time.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Linux as a Telephony Platform

David Sugar

Issue #43, November 1997

Using Linux as a base to implement advanced telephony applications.

In "Let Linux Speak" (*LJ*, January, 1997), I demonstrated some fun applications for the SPO256 text-to-speech board. Buried in that article was a brief discussion on the potential for using text-to-speech as a telephony resource and for using Linux as a telephony services platform.

Talking about "telephony services", or computer telephony in general, can mean many things. The history of computer telephony, while an interesting subject, is not our primary focus. Instead, I will discuss the use of Linux as a platform for voice response and path switching, for PBX integration and switch call control, and for the extension of traditional voice applications onto the Internet.

Telephony Overview

Voice response includes many applications such as traditional voice mail and interactive voice response (IVR) systems utilized as automatic "dial and survey" machines. These applications are typically built around multi-channel voice telephony boards that capture and play back digitized speech and that generate and listen for DTMF digits and call progress tones. The more advanced of these boards offer on board DSP resources, emulate FAX/modem services and perform speech recognition. The largest single vendor of this kind of board is the Dialogic Corporation.

PBX integration involves direct computer control of a PBX switching system. Many vendors have specialized boards and/or serial interfaces which run proprietary protocols for gaining access to different switch features. Generally, PBX integration is implemented as a telephony server or API, such as Microsoft TAPI or TSAPI (currently supported only under Novell Netware). Usually, these APIs implement first party call control for desktop applications (such as putting

a telephone image and dialer on a desktop, controlling a digital telephone directly as a “terminal device”) or as third party call control for server applications (such as ACD: automatic call distributors).

The whole area of Internet telephony is vastly interesting and intriguing. Most often, the first thing that comes to mind when one says “Internet telephony” are those nifty programs that allow computer users to place low-grade international telephone calls for free over the Internet. This same technology, when applied on a private corporate LAN with sufficient bandwidth, could provide a cheap means of inter-office switching (much like tie line services and expensive private T-1 networks) and a better solution for ACD agent positions.

Traditional low-cost telephony solutions have historically been implemented either under MS-DOS (with, perhaps, a custom real-time kernel) or under OS/2. The need for highly specialized real-time operating systems to drive multi-channel voice applications has disappeared as a result of increased CPU power, and equally important, the increased sophistication and power of add-on telephony boards. Many of these boards now manage I/O and call state largely on their own with only an occasional need for direct intervention. In the past guaranteed maximum interrupt latency was the mantra for evaluating real-time performance in complex voice processing systems; I now find support for real-time predictable scheduling policies more important.

A Look at Operating Systems for Telephony

While Windows NT is commonly touted as a “telephony operating system” these days, there are several serious considerations. The first is simply expense; a Windows NT machine means a machine with a video display. More often, telephony applications are deployed on dedicated stand-alone machines which sit in phone closets and, ideally, require only remote management.

Some of the same optimizations that make NT work better than Windows 95 as a desktop machine get in the way of using it for telephony applications, or for use as a telephony server and workstation concurrently. For example, one finds strange scheduling quirks that occur as NT optimized video drivers, which are now given the highest priority, update large areas of the screen.

Finally, even in today's world of cheap RAM, NT requires a minimum of 32MB, while Linux runs smoothly in 8MB or less. Even in the low-end commodity voice market, where MS-DOS-based voice mail systems predominate and a \$50.00 change in margins can make or break a product, these costs are very important. Now, imagine a \$2000 voice mail system, or, even better, a \$1000 voice mail “machine”, with desktop integration, multi-site networking and voice/email exchange, and what that system would do to the bottom of the commercial voice mail market.

Why not OS/2? Well, first there is always the question of “will it be around?” Second, the last non-desktop optimized release of OS/2 was 1.3, and it is still the most commonly used and supported release of OS/2 in voice processing products today. OS/2 driver support still exists in the voice response OEM marketplace, but it is not a rising star.

Why not DOS? Simply put, one cannot easily run network services from a DOS machine. In tomorrow's world, voice mail will have to present voice messages on the desktop, whether through proprietary means or through a web server and standard e-mail protocols. Other advanced user applications and networking services will need to be leveraged onto these once dedicated stand-alone voice processing machines.

And what about Unix? For many years, some variants of Unix have been used successfully in voice processing, typically in vertical market applications. The complete failure of the major Unix vendors to understand the CTI market and create appropriate software licensing terms or stripped down embedded releases have kept the cost of using these systems prohibitively expensive as a general purpose CTI platform. For example, a Unix machine for voice processing may not need NFS, many user utilities or X Windows. However, it does need sockets and a web server for administration and desktop telephony. No major Unix vendor seems to know how to properly license such a stripped down, embedded configuration.

So what are we left with? An inexpensive operating system capable of using inexpensive hardware, of running a mix of user and real-time scheduled processes, of remote management without the need for a local console, of integrated networking and of giving months of reliable service unattended. Only Linux and Free-BSD fit these criteria.

PBX Integration

As mentioned earlier, most vendors support some form of direct PBX integration. By far the most common method of integration, in the broadest sense, is “call detail”. Virtually every PBX on the market is capable of generating call detail, also available as serial data, that can be used for call accounting systems.

Call detail is often represented by different formats from different manufacturers, and may change within different releases of the same base product. The kind of information varies from simple information on in-bound/out-bound trunk usage (for example, number called, date and time, duration and extension called from) to complete call traces, whereby every aspect of a call's passage through a PBX is detailed.

Most PBXs support “programming ports”, which are essentially serial ports hooked up to a dumb terminal or PC running a terminal program like ProComm. These programming ports are typically connected to modems to allow remote maintenance of the PBX.

The programming port is quite useful and is often overlooked. From it one can find and change every aspect of a typical PBX system, such as hunt groups, directory number assignments, ringing and restriction patterns, forwarding, feature buttons on digital sets, etc. However, each piece of telephony equipment is different in the way that it handles programming. The Fujitsu F9600 has perhaps one of the easiest command languages to integrate, in that the programming interface is represented as a shell, with a shell-like command language and predictable response strings for both successful and failed commands. This makes it ideal for use with dumb serial terminals or automatic program control.

Some systems, such as the Nitsuko 384i, implement their program port as a menu system on the assumption that the tech maintaining the switch is on a form of ANSI terminal. Others that have a serial programming interface, such as the Panasonic DBS, behave and accept commands as if they were a digital telephone set, and are more unusual. These systems require a lot of work to be usable from an application standpoint.

One universal form of integration, though not widely implemented, is SMDI. SMDI is a protocol originally developed by Bell Labs for central office automated call response equipment. SMDI, when available, is commonly used for voice mail integration.

SMDI is carried on a serial port, and involves a simple, standardized protocol for sending commands and receiving messages from a PBX. Unfortunately, SMDI is pretty much limited to identifying the origin of a call, when it appears on a “message detail” extension (e.g., the extension of an attached voice mail system), and control of “message waiting” indicator lamps on phones. However, as this functionality is useful to basic voice mail systems, we will discuss SMDI again later in this article.

Systems that do not provide SMDI use tones for signaling and integration of third party voice mail systems. These tones are DTMF, and appear on the port(s) where the attached voice mail or IVR application handles its voice path. Since this form of integration is very limited and involves voice boards, I won't discuss it further.

This leaves us with a broad class of “other” integration ports. These are typically serial ports with a custom API or communication board. They use proprietary

protocols to indicate signaling and control activity as calls pass in to, or are generated out of, a telephone switch. The nature of these messages vary widely from one vendor to another.

Some vendors are very keen on providing support for third parties to develop call control applications. Harris, in particular, not only documents their primary serial control protocols (known as Harris HIL and WIL respectively), but also provide third parties with actual source code that demonstrates how to implement their protocols. Other vendors, such as Fujitsu, provide full technical documentation and specifications for their protocols to interested third parties. The English is a little poor in the Fujitsu documentation, but at least they make some effort to promote their interface (TCSI).

Alas, many vendors feel there is need to keep such information secret. Doing so has slowed the widespread development of general purpose computer telephony application interfaces and restricted development to a very limited set of players; namely, Novell TSAPI and Microsoft TAPI. This wall of secrecy limits most third parties from creating more advanced integrations around the strengths of a specific brand of equipment's feature set and permits the manufacturer to pick and choose who will be allowed to directly integrate their product.

Let's take a look at the Fujitsu F9600 PBX in detail. It includes a serial port for programming. Fault messages and system status and call detail also flow out of this port. It has an optional board with a synchronous 19.2Kbps TCSI interface known as TCSI. Considering the fact that it is a slow speed link, it seems unnecessary.

In addition, Fujitsu has their own telephony architecture built around a telephony server. This telephony server runs under SCO (ODT 3) Unix and provides a separate server for each port, as well as some application servers. A primary telephony server is used to control the TCSI interface through an Eicon HDLC/X.25 board. A second, separate server is used to control the programming port. Several applications can reside on the telephony server, each of which uses a separate server and communicates via TCP/IP sessions to the MML (PBX programming interface) and TCSI servers. In addition, an interface driver can be loaded under a separate machine running Novell Netware, along with Novell Netware Telephony Services, to provide a TSAPI interface to the PBX via the SCO machine.

The multi-tiered server structure Fujitsu uses enables applications to directly utilize TCSI services, the MML interface or a generic universal API set (TSAPI). However, it uses two separate machines and operating systems to build this

structure, and each low-level server is not directly aware of potentially useful information contained in the other.

Other vendors hide the low-level interface to the PBX completely and provide only a TSAPI (or TAPI) interface for their switch. These interfaces use only the information found on one port (the call control port) and ignore information available on the programming port and the potential to create remote network administration through the universal API.

In fact, the universal API concept as expressed in TAPI and TSAPI is based on creating a model PBX call control system as a server. A single API and single protocol is used between the application and this universal server. The server is then implemented by providing a lower level driver (service provider interface) to figure out how this generic PBX model applies to the peculiarities of the vendor's specific hardware and call control model.

The single-model API fails in one of two ways. First, some features of the model API are not available under a specific vendor's low-level SPI driver due to the inability to represent and recreate specific functionality within the universal call control model. Second, really neat and unique features the vendor may support beyond the basic universal model are unavailable unless, like Fujitsu, a second direct protocol server is provided.

To solve these and other problems in advanced integration for free and open systems, I completely abandoned the universal API and single protocol hierarchical model used in TSAPI and TAPI. I feel a better solution is a multi-protocol "TServer" used as a single unified server and network point of entry for access to *all* services that can be made network accessible from a telephony switch:

- generic call control interfaces (such as a TAPI or TSAPI call server)
- programming interfaces
- network SMDI (either directly from the switch or emulated through the call control interface)
- call detail recording

The Panasonic DBS

The Panasonic DBS is a widely used digital Hybrid Key system (not to be confused with the Panasonic KXT). The choice of creating a model server around the DBS was predicated on both immediate familiarity with the internals of this switch and availability of hardware on which to test.

The DBS has two interfaces that can be used. The first, a built-in serial port, is used for **both** programming and call detail recording. For this reason, the server must be sensitive to the current “operating state” of the port (i.e., whether it is currently being used for programming). As mentioned earlier, commands to the DBS programming port are represented in a manner analogous to the way one would program the switch from a digital key telephone.

To simplify the DBS serial programming for application use, I created a simple protocol to send the “change value”, “get value” and “clear value” requests which make up the conceptual heart of the operations used for programming the DBS from a telephone. These commands, and the specification of an address for the values being affected, make up the essential elements of my DBS programming protocol.

The DBS also includes an optional API card. This card has a serial port used for call control and switching. This low-level protocol is unidirectional and involves a message packet and access negotiation protocol. The interface is serial, with no provision for hardware flow control. Messages are essentially exchanged between the host machine and the switch, similar to Fujitsu TCSI, Harris HIL or other equipment. Later versions of the DBS API include two serial ports, one of which has extended features to support a Netware TSAPI server.

The Linux TServer

With all these protocol interfaces to maintain and the need to connect a single point of network entry, a monolithic server cannot be used. Since the points of connection (API interface and programming interface) can be used as a shared resource by multiple applications simultaneously, a simple “fork()ing” server common in Unix is inappropriate. Instead, there is a set of protocol servers connected to a single “traffic cop”. This multi-protocol traffic cop binds itself to a single network port address providing universal access from a single network session and acts as the primary TServer.

In the simplest approach, each protocol (interface) server is run as a child process using anonymous pipes. The TServer accepts and manages user application sessions. Anonymous pipes for the protocol server are very fast and require low overhead, especially with Linux.

In the DBS are several layers of servers for the API portion of the interface. This layering behaves a little like a stream driver stack, and it could probably be implemented as such under a streamable Unix.

At the lowest layer is the DBS packet link driver (**papild**). This driver negotiates the unidirectional packet exchange with the DBS and normalizes DBS messages

to a single fixed packet. The reason for a fixed size packet is to allow transfer of all messages to occur between the different protocol layers by performing atomic **read()** and **write()** calls. This method allows clean tracking of "split" messages on a non-blocking pipe that has been overwritten past its buffer.

The DBS packet channel state driver (**papicsd**), is a second stream layer built from pipes above **papild**. This layer is aware of the peculiar nature of the DBS wherein all message events are initiated with regard to one or more channels, which are virtual station ports associated with the DBS API card. Requests that may generate reply messages are queued within **papicsd**. If a request with an expected reply fails, the **papicsd** layer can generate timeout events. Similarly, the **papicsd** layer is responsible for glare resolution, a situation that occurs when attempting to take a station *off hook* at the same time that an incoming call occurs. Finally, the **papicsd** layer can trap ridiculous or meaningless commands, such as hanging up a port that is already known to be on hook and returning generated error messages back to the application.

By having a separate process for **papild** and **papicsd**, they become easier to write and debug. Also, since the DBS has no flow control, **papild** can be dedicated to I/O specific operations and, perhaps, even run within real-time scheduling constraints in Linux (2.0 and above). This, in conjunction with the non-blocking atomic pipe as a means to drive packets up and down the protocol layers, requires less overhead and works much more smoothly than some of Panasonic's original QNX application products for the DBS.

On the programming interface (labeled SMDR on the DBS), a single process is used. When no programming requests are active, it simply spools CDR data for call accounting purposes. When a programming request occurs, the server changes state to a programming port interface server and remains in that state until 30 seconds pass without any program requests being made.

In addition to the TServer itself, there is one more protocol layer on the papi stack. This utility layer maps DBS API channel states to station card and trunk states based on the live DBS API data. This mapping layer, **papiltu**, is essentially a SPI-like layer, in that it represents the DBS's current state in an intermediary form in terms of line and trunk activity that may make more sense than the DBS API channel architecture. This layer also holds the DBS station directory map, which is learned during initialization from the programming interface. The information it maps and maintains is shared with other DBS related protocol servers (such as the SMDI server) through a shared memory block.

By using the programming port and **papiltu**, the papi protocol stack is able to automatically configure itself by reading current DBS programming without user application programming and setup. This replaces the clumsy manual

approach of a “channel configuration” screen to separately program the API port configuration, as used in earlier Panasonic application products. The **papiltu** functionality was originally an integral part of the primary TServer application server, but with the restrictions on publication of DBS API material, it was made into a separate layer to facilitate publication of the primary TServer source code, if none of the papi stack sources can be published. This matter is still under discussion. (See comments at the very end of article.)

The TServer line/trunk state maps actually make the TServer behave a little like the SPI layer of TAPI or TSAPI, as it represents an intermediate form between a generic PBX model and the actual physical model. In this case, in addition to driving direct hardware interface protocols, our TServer can communicate with logical protocol drivers which are not directly connected to any physical hardware.

One such logical protocol driver is the DBS SMDI driver. Connected through anonymous pipes, like the **papicsd** and **smdr** layer drivers, the **smdi** protocol driver talks to the shared **papicsd** by going back through its connection to the TServer. Since it does not depend on a single shared hardware resource, as does a real (physical) driver, an instance of the SMDI driver is created (forked) for each client application. This driver inherits the TServer socket accepted for the client connection and the pipe back to TServer; hence, communications between the SMDI emulation driver and the client application is always direct (i.e., it has no TServer routing overhead).

Based on the SMDI example, one can easily create protocol modules and matching application interfaces to fully emulate TSAPI or TAPI services directly. One can also talk to a vendor-specific hardware protocol directly (such as the papi layers or SMDR interface) or create entirely new arbitrary integration protocols. The TServer, as a single point of contact traffic cop, allows the client to perform authentication and to select the interface protocol that the client application wishes to use.

TServer Applications

An interesting feature of the DBS is the ability to turn a large display telephone into a simple kind of terminal device. This allows the display content to be controlled by the application, and the application to receive input events when keys are pressed on the telephone. Another feature is a special “hot key” (also known as the ACD key), that functions as an “attention” key that generates an API event when pressed, regardless of the current telephone state.

One of the first DBS applications I created is a simple menu program for attaching “applets” to a telephone. When the attention key is pressed, a simple menu of applications appears from which one can be selected. One such

application is used to immediately show status information for my server (how many users on-line, uptime, etc.) along with a soft key menu item to force a server reboot.

Another digital telephone application of mine is a more advanced speed dialer that has no capacity limit and is programmable from the telephone. This application resembles the Fujitsu "Dial-by-Name" server application in concept. The DBS has its own internal speed dialing directory. Since alphanumeric text is hard to enter through the phone, I wrote a simple Visual Basic program to connect to the SMDR programming protocol in order to program DBS speed dialing.

A possible future application that comes to mind is empowering users to program their own phones from the desktop or perhaps from web pages.

Conclusion

Many opportunities exist for the use of free and open systems in computer telephony, especially for those telephone vendors wise enough to expand their marketing opportunities by allowing third parties to freely address issues and applications beyond their own immediate scope. While I choose to use the Panasonic DBS and, with it, have accepted restrictions on disclosure and source publication, several other vendors have expressed interest in having their equipment featured in a follow-up article.

When I started this article, I became aware of the effort to create a standard and open Internet protocol for telephony integration, known as "stp", Simple Telephony Protocol. After some debate, I have chosen to fully embrace stp, and the current software described in this article is being rewritten to support and comply with the evolving stp standard. The name "SwitchLink" has also been adopted for it (swilink for short). My intention is that swilink will become widely available as a free and open software package for release with all mainstream Linux distributions. Thus, free and open telephony will become the norm rather than the exception for Linux.

Recently, there has been considerable change in the attitudes of several key hardware vendors in the telephony business with regard to Linux. I now believe opportunities to write on the use of Linux as a general purpose and high performance Internet telephony platform may be possible much earlier than I anticipated.

Glossary



Best known for WorldVU, a public BBS system for Linux, **David Sugar** is currently employed as Director of Software Engineering for Fortran Corp. and uses Linux for commercial telephony development. He maintains his own Internet server under Linux, and may be reached for comment via <http://www.tycho.com/>. He can be reached via e-mail at dyfet@tycho.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Game Control Design

Dave Thomson

Issue #43, November 1997

A few simple design guidelines are presented to make your games more enjoyable to the players.

Sweets are first tasted by the eye, but flavour is the heart and soul of all confectionery. --John Millar (1826-1896)

It's called a joystick because games are fun—or at least they're meant to be. Of course, I doubt this is the real reason. Nevertheless, I stand by my reasoning as the definitive one. It may not be true, but it makes sense.

Although there are several books available on the market about game programming, there are very few about game design. The few that are available tend to be either very brief and shallow, or they're not really geared towards video game design, instead covering the intricacies of board or paper games.

To me, it seems obvious that there is little point in writing a game that no one wishes to play. There are certain guidelines that can be followed in order to make your game more appealing.

Graphic and sound content obviously play a big part in making your game enjoyable, but these areas are covered amply elsewhere. This article covers the lesser known basics of playability: the control system and level design. Along the way, I'll make the occasional reference to a D*S game I wrote about a year ago called *Teardrop Explodes*. If you wish to try it out, feel free to download it from <http://www.geocities.com/SiliconValley/Heights/2276/>.

The Control System

In any game the control system should be kept simple. Why use four different buttons when one will do? An example of this syndrome appears in most of the

new 3D soccer games. Here, 3 or 4 buttons are used just to kick the ball in a variety of manners (pass, shoot, etc.). Older soccer games used only one button. Why? Well, that's all that they had, and the action performed was dependent on how long you held the button down. Perhaps this is why the so-called "next-generation" fail to match the playability of the older titles.

Similarly, if the control system isn't intuitive, players will not enjoy playing your game. One simple reason that may happen is that they may not be able to use the keys you chose for the game; therefore, your game should allow the user to redefine the controls. Incidentally, I failed to follow this practice in *Teardrop Explodes*. I later discovered that it was not playable with keyboards on which the cursor keys are laid out differently than my own—a lesson learned the hard way.

Another reason for this is that the controls go against the player's expectations. For example, I recently played a game where I used a joystick to control a spaceship. I naturally expected the craft to bank when I pulled the stick left or right—wrong—instead, this action caused the craft to rotate. Needless to say, I didn't enjoy playing that game.

Level Design

Arguably, this is the single most important section of any game production. The player has to be drawn into the game, and immersed in the atmosphere. There are various techniques to gain his interest, such as lighting and ambient sounds. However, it is also the most difficult area to find any guidelines, since all games differ in their level structure.

Ambient sounds can be used for such trivial things as a pipe dripping or the growls of monsters. These sounds should be used with care; otherwise you could end up swamping the player and ruining his game experience. If at all possible, they should be used only when other sounds are not being played. One way to do this is to use "trigger points" in the level, which are similar to invisible tiles that play the required sound when the cursor passes over them.

The player should be surprised once in a while. For example, how many times did you jump the last time you played *Doom*? There you are, cautiously stalking along a corridor, when suddenly a crowd of demons charge through a hidden door behind you—this is exactly the kind of effect you should aim to achieve.

You should also allow the player to interact with the environment. If the player sees or tries something he can't get or do, he becomes frustrated. In arcade games, this generally involves wanton destruction of the surroundings. In the case of your game, let it happen in some fashion. Even if the scenery doesn't

actually change, use big explosions—big explosions always satisfy a player's thirst for destruction.

Finally, and this is the big one, play test. Play testing your level designs and game in general is very important. Things to look out for include the general “flow” of the game, difficulty levels and the balance of the control system. I talked about control systems earlier, so let's consider the other two points.

In order for a game to “flow”, all the disparate elements of your game should come together and mesh seamlessly. This is a harder objective to achieve than it sounds, so it helps to decide on a common theme for the game. You may want to come up with an intricate story line or a particular graphical style. Whatever you choose, just make sure that it remains consistent—that nothing seems out of place.

The difficulty setting of a game is, again, quite hard to balance exactly right. Obviously, the game should start easily and get progressively harder. What's the best way to go about this? Well, the most effective method I have come across is to have a fairly simple starting section for the game, which allows the user to experiment with the control system and use different abilities of their on-screen persona. After this, you can throw some harder game elements at them. However, don't be relentless in this action; slightly. The easiest way to do this is to have a “pause game” option. You could also have bonus levels or hidden games. The best way to judge where to use one of these lulls is, of course, to play test.

Bits and Pieces

There are other things to consider regarding playability. For example, in *Teardrop Explodes*, I tried to create the hi-score table so that players would find it fairly easy to get their name in lights. On the other hand, I made it quite hard, but certainly possible, to beat the highest score. You should also save the table, as this gives players something to beat after they have completed the game.

Saving the game's configuration data is also advisable. The player should have to monitor brightness levels only once, for example.

A recurring trend in games is to use cut-scenes to further the story line. If you use them, try to keep them fairly short and allow them to be skipped through as well. The same goes for the “Game Over” sequence, where the player will (hopefully) be eager to have another try at your game. Also, if you have a “lives” system, don't take too long in restarting the game. If you do, the player might get frustrated and just switch off.

As for game completion, you should offer something that rewards the player directly for how much effort he has put into the play. Someone who spends three months of his life trying to finish your game will want something fairly substantial. If it takes 30 minutes, on the other hand, a single static screen would suffice.

Again, I'm not claiming that following these tips will make your game a block buster. Guidelines such as these are purely optional; if you disagree with anything, feel free to do it your way. Incidentally, the quote at the start of this article is by a Scottish sweet-maker and appears on the back of his company's wrappers. I include it because I think the quote can be applied equally to games, particularly in these days of eye candy over content. I wish you luck with your projects, and I look forward to seeing a flurry of games activity on the Linux scene.

Dave Thomson is close to graduating with a CS degree from Heriot-Watt University, in the UK. He can be contacted at gameskitchen@geocities.com for heated debate on the virtues of almost anything and, in particular, games.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Linux and the Alpha

David Mosberger

Issue #43, November 1997

Part 2 brings us optimization techniques for speeding up code to get the best performance from your Alpha or other RISC processor.

In this article, I will discuss techniques to optimize code for platforms running Linux on Alpha processors. It is based on four years of experience with the Alpha architecture. The primary lesson from this experience is that, for many applications, the memory system, and not the processor itself, is the primary bottleneck. For this reason, most techniques are targeted at avoiding the memory system bottleneck. Since the gap between processor and memory system speed is large, these techniques achieve performance improvements of up to 1700%. While the focus is on the Alpha architecture, many of the covered techniques are readily applicable to other RISC processors and even modern CISC CPUs.

The tricky part in discussing how the same techniques work on other architectures is that we want to do this without igniting a war over which CPU architecture is the “best” or the “fastest”. Such terms are, to a good degree, meaningless, since they can be applied usefully relative to a given problem only. For this reason, performance results are presented as follows: for the Alpha we present both absolute and relative results. The absolute numbers are useful to give a concrete feel for how fast the code is. The relative results (i.e., speed ups) are what tell us how well a given technique works. Where meaningful, we also list the speed up (but not the absolute performance) achieved on a Pentium Pro-based system. Only listing speed up for the Pentium Pro case makes it impossible to tell which system was faster on a given problem while still allowing us to compare the relative benefits. (To avoid any misconception: this arrangement was not chosen because the Alpha performed poorly; the author has been using Alphas for some time now and is generally pleased with the performance level they achieve.)

Since an architecture per se doesn't perform at all, let us be a bit more specific about the systems used for the measurements:

- Alpha system: The Alpha system was an AlphaStation 600 5/333 (aka Alcor). It has a 333MHz 21164 processor with 4MB of third-level cache and 64MB of main-memory. While a nice (and very expensive) box, it is by no means the latest and greatest of the available Alpha systems. At the time of this writing, much faster and much cheaper 500MHz systems have already been around for a while.
- Pentium Pro system: The x86 system was a Gateway 2000 with a Pentium Pro processor with 32MB of main memory and 256KB of second-level cache. For clarity, this system is referred to as "P6" during the remainder of this section.

Both the Alpha and the P6 systems were running Red Hat 4.0 with kernel version 2.0.18. The compiler used was **gcc** version 2.7.2. On the Alpha, option **-O2** was used (with this version of gcc, using an option setting of **-O3** or higher generally results in slower code.). On the P6, options **-O6** and **-m486** were used.

It is also illustrative to compare gcc to commercial Alpha compilers, such as Digital's GEM C compiler. The GEM C compiler usually generates somewhat better code but now and then it creates code much faster than gcc's code (this usually happens on floating-point intensive code). For this reason, some of the measurements also include the results obtained with GEM C. This compiler was invoked as:

```
cc -migrate -O4 -tune ev5 -std1 -non_shared
```

It's not clear which version of the compiler it was—it came with Digital UNIX version 3.2.

Avoiding Integer Division

The Alpha architecture does not provide an integer division instruction. The rationale for this is:

1. Such operations are relatively rare.
2. Division is fundamentally of an iterative nature, so implementing it in hardware is not all that much faster than a good software implementation. (See Reference 1.)

Nevertheless, there are important routines that depend on integer division. Hash-tables are a good example as computing a hash-table index typically involves dividing by an integer prime constant.

There are basically two ways to avoid integer division. Either the integer division is replaced by a floating-point division or, if the division is by a constant, it is possible to replace the division by a multiplication with a constant, a shift and a correction by one (which isn't always necessary). Floating-point division may sound like a bad idea, but the Alpha has a very fast floating-point unit, and since a 32-bit integer easily fits into a double without a loss of precision, it works surprisingly well. Replacing a division by a constant with a multiplication by the inverse is certainly faster, although it's also a bit tricky since care must be taken that the result is always accurate (off-by-one errors are particularly common). Fortunately, for compile-time constants, gcc takes care of this without help.

To illustrate the effect this can have, we measured how long it takes to look up all symbols in the standard C library (libc.so) using the ELF hash-table look-up algorithm (which involves one integer division by a prime constant). With integer division, roughly 1.2 million look ups per second can be performed. Using a double multiplication by the inverse of the divisor instead brings this number up to 1.95 million look ups per second (62% improvement). Using integer multiplication instead gives the best performance of 2.05 million lookups per second (70% improvement). Since the performance difference between the double and the integer multiply-by-inverse version isn't all that big, it's usually better to use the floating-point version. This works perfectly well, as long as the operands fit in 52 bits, and avoids having to worry about off-by-one errors.

Keeping the Translation Lookaside Buffer in Mind

Two parts in modern CPUs that are easy to forget are the data and instruction translation lookaside buffers (TLBs). The TLBs hold the most recently accessed page table entries. TLBs are necessary since it would be far too slow to access the page tables on every memory access. Since each TLB entry maps an entire page (8KB with the current Alpha chips), the TLB is usually not the limiting factor to performance. The catch is that when a program does suffer from excessive TLB misses, performance will go down the drain fast. In such a case, the slowdown can easily be big enough that it is worthwhile to switch to a completely different (normally slower) algorithm that has a better TLB behavior. For example, on the Alpha system reading one word from each page in an array of 63 pages (504KB) takes about 15ns per access. But doing the same in an array of 64 pages takes 65ns per access—a slow down of more than a factor of four. Since the second-level cache in that system is 4MB in size, this jump in access time is purely due to the data TLB.

The TLB is not usually a first-order bottleneck but a small experiment did show that a hash table that is 50% full and exceeds the data TLB size is no faster than a more compact search tree that needs two memory accesses per look up (the

hash table needs just one memory access, but since it exceeds the TLB size, that one access is slow). In general, the TLB may be the primary bottleneck when large data sets are accessed more or less randomly and sparsely.

Avoiding Memory Accesses

On modern systems, memory accesses are bad and computation is good. In the ideal case, we would like to completely replace memory accesses by computation. This obviously is not always possible; but where it is possible, the payoff can be big.

For example, let us consider the problem of reversing all the bits in each byte in a long integer. A byte is reversed as follows: bit 0 is swapped with bit 7, bit 1 with bit 6 and so on. Since we want to reverse all the bytes in a long, this algorithm is applied once for each byte in the long.

Why would anyone want to do this? As you may know, both the Alpha and the x86 architecture are little-endian, but when IBM designed the VGA graphics adapter, it chose to use a big-endian bit order for the pixels in the graphics memory. That is, bit 7 in a byte corresponds to the left-most pixel and bit 0 to the rightmost. This is backwards since the coordinate value for the left-most pixel is smaller. So, byte reversal is indeed a relatively important operation for VGA X servers.

The traditional way of implementing byte reversal is shown by the code in [Listing 1](#). To conserve space, we show the code to reverse a 4-byte integer only. The 8-byte long case is a straightforward extension of this one. The code assumes that array **byte_reversed** has been initialized such that **byte_reversed[i]** is the reversed value of **i**. With this naive algorithm, each byte reversal involves a table look up.

All listings in this article are available by anonymous download in the file <ftp://linuxjournal.com/pub/lj/listings/issue43/2487.tgz>.

Since the Alpha is a 64-bit architecture, this means each long reversal involves eight memory accesses. How could we avoid these expensive memory accesses? Well, a simple and arguably more intuitive way to implement byte reversal is to use shifting and masking to swap the bits. The code that does this is shown in [Listing 2](#).

Note that, except for the initialization of **mask**, this code is completely independent of the width of a long. So, to make this work on a 32-bit machine, all we need to do is initialize **mask** with 0x01010101 instead.

Now let's see how the implementations compare. Table 1 gives the results for the Alpha, the Pentium Pro (P6) and a 120MHz Pentium Notebook (P5). In addition to the two implementations shown above, the table also includes a row that shows the performance of the naive implementation when coded in x86 assembly code (implementation `byterev_x86`). This routine comes straight from the XFree86 v3.2 distribution.

Table 1. Comparison of Byte-Reversal Routines

Implementation	Alpha		P6	P5
	abs	rel	rel	rel
<code>byterev_naive</code>	55MB/s	1.00	1.00	1.00
<code>byterev_long</code>	72MB/s	1.31	1.13	1.17
<code>byterev_x86</code>	n/a	n/a	0.56	1.51

As the table shows, the byte-reversal routine that avoids memory accesses is over 30% faster on the Alpha. Interestingly, on the P6 this routine is also the fastest. The benefit there is less (only 13%), but given that it's more portable and more intuitive, there is no reason not to use it. What's stunning is the complete failure of the assembly version on the P6. That routine is only half as fast as the version that avoids memory accesses. This may be due to the assembly routine's extensive use of byte accesses to CPU registers. For the Pentium notebook, as shown in the P5 column, the assembly-code routine is the fastest version. Don't be misled by the relative performance numbers; in terms of absolute performance, the P5 is just half as fast as the P6.

To summarize, not only is `byterev_long()` by far the fastest version on the Alpha, it also appears to be the right solution on a P6.

Lending gcc a Hand

One reason gcc sometimes generates significantly less efficient code than Digital's GEM compiler is that it does not perform inter-procedural alias analysis. What this means is that gcc's alias analysis is sometimes unnecessarily conservative. To illustrate the problem, consider the function shown in [Listing 3](#). It is a simple unrolled loop that reverses all the bytes in array `src` and stores the result in array `dst`.

The problem with this code is that the C compiler has no way of knowing how the `src` and `dst` pointer relate to each other. For all it knows, `dst` could point to the second element of the array pointed to by `src`. When this happens, the two pointers refer to overlapping regions of memory and they are said to alias each other. For the compiler, it is important to know whether the two pointers overlap, since that determines the degree of freedom it has in reordering instructions. For example, if the regions overlap, then storing a value to `dst[i+0]`

may affect the value of `src[i+1]`. Thus, to be on the safe side, the compiler must generate the loads and stores in the above function strictly in the order in which they occur in the source code.

Now, if it is known that the two arrays passed to the function never alias each other, we can lend gcc a hand by explicitly giving it this information. We can do this by first reading all the values from the memory, then doing all the computation and finally storing the results back to memory. Thus, the above code would be transformed into the code shown in [Listing 4](#).

Since all of the stores occur at the end, gcc knows immediately that none of the stores can affect any of the preceding loads. This provides it complete freedom in generating the best possible code for the loads and computation (the assumption here is that `byterev_long` is an in-line function).

On the Alpha and most other architectures with lots of registers (e.g., most RISCs), this kind of code never (or at least very rarely) hurts performance and usually improves performance for gcc. Unfortunately, the same isn't true for the x86 architecture. The problem there is that only a few registers are available. So, the code that's better for the RISCs is usually worse for the x86 due to additional stores and loads that are necessary to access the temporaries that end up on the stack. Performance numbers for the two versions are given in Table 2.

Table 2. Performance Values of Byte Reversal Source Code

Implementation	Alpha		P6
	abs	rel	rel
<code>vecrev_naive</code>	82MB/s	1.00	1.00
<code>vecrev_sep</code>	93MB/s	1.13	0.73

As the Table 2 shows, merely by separating loads from computation from stores gains 13% with gcc on the Alpha. In contrast, the same code loses 17% on the P6. This illustrates that while most of the techniques in this paper apply to both architectures, there are important differences that sometimes necessitate different coding to achieve the best performance.

Improving the Memory Access Pattern

Now, let's consider a simple problem: matrix multiplication. While simple, it is typical of a problem that is considered floating-point intensive. In reality, most floating-point intensive problems are also memory intensive. For example, they process large vectors or matrices. Thus, the memory access pattern often plays a crucial role in achieving the best possible performance. The textbook implementation of matrix multiplication looks is shown in [Listing 5](#).

Here, the matrix pointed to by **a** gets multiplied by the matrix pointed to by **b** and the result is stored in **c**. The matrix dimension is passed in argument **dim**. On the Alpha, a 512 by 512 matrix multiply with this routine executes at about thirteen million floating-point operations per second (MFLOPS). This is not too shabby, but let's see whether we can squeeze more out of the machine. Having learned our lesson in performance optimization, we might try to unroll the inner loop and avoid all multiplications due to indexing. This does indeed result in a faster version: now, matrix multiply executes at about 15 MFLOPS.

gcc's optimizer is unable to eliminate the induction variables and, hence, the multiplications due to indexing. **gcc** 2.72.1 and earlier have a bug in this area that appears when generating code for the Alpha. If the index variables are declared as **long** instead of **int**, **gcc** is able to eliminate the induction variables, as one would expect.

Rather than declaring the problem solved, let's think about the memory access pattern for a minute. Each element in the result, matrix **c**, is a dot product of a row in **a** and a column in **b**. For example, the element **c[0][0]** is computed as the dot product of the first row in **a** and the first column in **b**. This is illustrated in Figure 1. In our naive matrix-multiply routine, this means that the accesses to **a** form a nice, dense, linear memory-access pattern. Unfortunately, things do not look quite as good for **b**. There the memory access pattern is sparse: first, the element at offset **0** is read, then the one at offset **dim** and so on. Such sparse access patterns are bad for many reasons. Suffice to say it's easiest to optimize a machine for dense, linear accesses, so it is likely that those accesses will always be the fastest ones. Fortunately, there is a simple trick that avoids the bad access pattern for matrix **b**: before doing the actual matrix multiply, we can simply transpose matrix **b**. Then, all memory accesses are dense. Of course, transposing **b** causes extra work, but since that matrix is accessed **dim** times, this may well be worth the trouble.

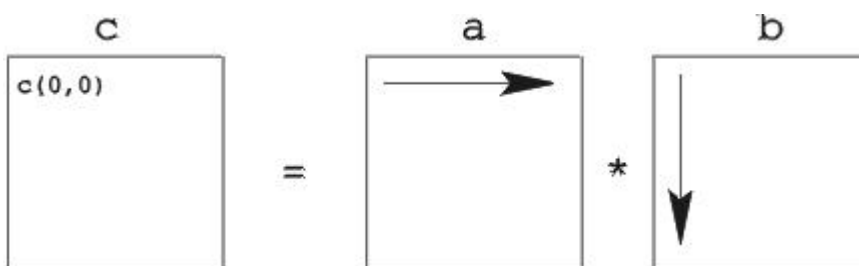


Figure 1: Matrix Multiply

So, let's change **matmul0** into **matmul1** by adding a matrix transposition in front of the main loop. The code in [Listing 6](#) assumes that **tb** is an appropriately sized temporary variable to hold the transposition of **b**.

If you thought 15 MFLOPS is fast, think again: `matmul1` executes at a blazing 45 MFLOPS. Next, we'll add some loop unrolling, etc. For `matmul0`, this bought us 25%, which isn't bad at all. If we unroll the loop eight times and do some other straightforward optimizations, we get the code shown in [Listing 7](#). For compactness, it assumes that `dim` is an integer multiple of eight. Was it worth the trouble? By all means yes: `matmul2` clocks at a full 80 MFLOPS. Whether you like this kind of code or not may be a matter of taste, but it is certainly fast.

Table 3 presents a summary of the performance results. For comparison, it also includes the results obtained when compiling the same code with Digital's GEM C compiler and the relative results for the P6. As this table shows, with `gcc` we achieved a performance improvement by over a factor of seven. More importantly, the optimizations paid off in all three cases. In fact, with Digital's compiler, the improvement was more than a factor of eight. The performance gap between Digital C and `gcc` is rather large. For integer code, the gap is usually smaller, but it certainly looks as if `gcc` needs some work in the floating-point area. Finally, notice that even on the P6 performance increased by almost a factor of four. This is encouraging since unrolling the loop eight times is a tad aggressive for the x86 architecture (because relatively few registers are available). Presumably, the code could be sped up some more, but the point here is that all three cases benefit from memory access optimizations in the same way.

Table 3. Performance Results of Matrix Multiply Routines

Implementation	Alpha				P6
	gcc		GEM C		
	abs	rel	abs	rel	rel
<code>matmul0</code>	11MFlops	1.00	13MFlops	1.00	1.00
<code>matmul1</code>	47MFlops	4.27	66MFlops	5.08	2.53
<code>matmul2</code>	80MFlops	7.27	105MFlops	8.08	3.97

Data-parallel Processing: MPEG Core Loop

Multimedia applications are the rage these days. All mainstream CPU architectures (with the notable exception of the PowerPC) have so-called multimedia extensions. The Alpha is ideally suited for such applications since it has been a 64-bit architecture right from the start. In fact, the Alpha multimedia extension is completely trivial; it adds only four new instruction types (vector minimum/maximum, pixel error, pack and unpack). Since some of these instructions can operate on different data types (byte or word, signed or unsigned), the total number of instructions added is 13, which is much smaller than the corresponding number for other architectures. The Alpha got away with so few additions because its original instruction set already contains many of the instructions needed for multimedia applications. For example, there is an instruction that allows eight bytes to be compared in parallel—a seemingly

simple instruction that can prove surprisingly powerful in a number of applications.

We illustrate this using `mpeg_play`, the Berkeley MPEG decoder. (See Reference 2). Since there is not enough space to illustrate all the optimizations that can be applied to this program, we'll focus on one of the most important operations in the MPEG decoder. This operation involves computing the average of two byte vectors. This is a frequent operation since video often contains images that can be represented as the average of an earlier and a later image. The straightforward way of averaging a byte vector is shown in [Listing 8](#).

This loop executes at about 94ns per byte average (iteration) when compiled with `gcc`. Unrolling this loop twice and reading ahead to the input values needed in the next iteration yields code that is probably close to optimal with this byte-oriented approach. Indeed, with `gcc`, performance increases to about 60ns per byte average. (Let's call this unrolled version of the function **`byte_avg1`**.)

To get even higher performance, we need to be a bit more aggressive. Considering that the Alpha is a 64-bit architecture, we would like to calculate the average of eight bytes in parallel. Reformulating byte-oriented algorithms in such a data parallel format is often trivial. For byte averaging, it's not quite as simple. The straightforward implementation requires nine bits of precision, since $255 + 255 = 510$. If we pack eight bytes into a 64-bit word, there is no extra ninth bit. How can we get around this? Obviously, we can divide the operands by two before adding them. That way, the sum is at most $127 + 127 = 254$ which conveniently fits into eight bits. The catch is that the result may be wrong; if both operands are odd, it will be one too small. Fortunately, it's easy to correct for this: if bit 0 in both operands is set, a correction by one is necessary. In other words, we can make space for that extra ninth bit by using an additional long register that is used to hold the correction bits. Since all intermediary results now fit into eight bits, the obstacles to a data-parallel implementation of byte averaging have been removed.

The resulting code is shown in [Listing 9](#). For simplicity, it assumes that the input vectors are long aligned and have a size that is an integer multiple of the size of a long. Note that macro `VEC()` takes an eight-bit value and replicates it once for each byte in a long—it's much more convenient and less error-prone to write `VEC(0x01)` instead of `0x0101010101010101`. Maybe it's helpful to explain the core of the averaging a bit. Variable `CC` holds the correction bits, so it's simply the bitwise AND of vectors `A0` and `B0`, masked with a vector of `0x01`. We divide `A0` and `B0` by two by shifting them to the right by one position and masking the resulting long with a vector of `0x7f`. This masking is necessary since otherwise bit 0 of the byte “above” a byte would sneak in and become bit 7 of that byte,

causing gross errors. The average is computed by simply adding the vectors A0, B0, and CC. This addition does not cause any overflows since, per byte, the largest possible value is $127 + 127 + 1 = 255$.

Despite its look, this code is actually very portable. For a 32-bit architecture, all that needs to change is macro VEC (and even that is necessary only to get rid of compiler warnings). Byte order is not an issue since even though the data is accessed one long at a time, each byte is still processed individually. This data-parallel version of the byte-averaging loop runs at 5.3ns per byte-average—more than an order of magnitude faster than the unrolled loop.

A summary of the three averaging routines is given in Table 4. The relative performance is in terms of throughput (number of byte-averages per second) since that's both more intuitive and more impressive. Results for the Alpha are presented both for gcc and Digital's GEM C compiler; as usual, for the P6, gcc was used.

Table 4. Performance of Averaging Routines

Implementation	Alpha				P6
	gcc		GEM C		rel
	abs	rel	abs	rel	
byte_avg0	94ns/avg	1.00	69ns/avg	1.00	1.00
byte_avg1	60ns/avg	1.57	56ns/avg	1.23	0.82
byte_avg2	5ns/avg	18.80	4ns/avg	17.25	1.93

Note that GEM C generates much better code for the stupid version (byte_avg0) but just slightly better code for the clever version (byte_avg2). This is a common theme, at least for integer code: for well-structured code, gcc usually generates code that is on par with the GEM C compiler. The other interesting result is that the read ahead and loop-unrolling hurt on the P6. This means that byte_avg2 probably could be optimized more for the P6 (since it uses read ahead and loop-unrolling, too), but even so the P6 is twice as fast with the data-parallel version. This is impressive since the relative overheads are much higher for a 32-bit chip (the Alpha can amortize all the masking and shifting over eight bytes, whereas a 32-bit architecture has only four bytes).

How does all this affect performance of mpeg_play? This is best illustrated by comparing the original Berkeley version with the one optimized using the techniques described in this section (particularly data-parallel processing and avoiding integer divisions). (See Reference 3.) Comparing MPEG performance is a bit tricky since a large fraction of the time is spent displaying images. This can be factored out by using the option `-dither none`, which has the effect that nothing gets displayed (while the MPEG stream is still decoded as usual). Table 5 shows the result for this mode as well as when using an ordered dither. The

ordered dither itself was also optimized using data-parallel processing, which resulted in a version called **ordered4**. The options used for mpeg_play were -**controls**<\!s>**none**<\!s>-**framerate**<\!s>**0**<\!s>-**dither**<\!s>**D**. The value of **D** was either **none**, **ordered** or **ordered4**, as indicated in column labeled "Dither" in Table 5. The movie that was used in these measurements was a 320 by 240 pixel-sized computer animation called RedsNightmare.mpg. (See Reference 4.)

Table 5. MPEG Performance Values

Version	Dither	Framerate
Original	none	45.3 frames/sec (1.00)
Optimized	none	139.8 frames/sec (3.09)
Original	ordered	26.9 frames/sec (1.00)
Optimized	ordered	58.9 frames/sec (2.19)
Optimized	ordered4	98.2 frames/sec (3.65)

As the table shows, the optimization techniques do indeed result in tremendous performance improvements even at the application level. Of course, few people would enjoy watching a movie at 98 frames per second, but with the optimized code, you can either watch much larger videos in real-time, or you could have CNN on while watching your favorite movie. Who needs picture-in-picture capability when we've got real windowing systems?

Conclusions

The Alpha architecture is designed for performance and its implementations do indeed make for very fast systems. Since its chips run at very high clock frequencies, the Alpha usually benefits the most from simple techniques that improve the memory-system behavior of a given program or algorithm. A few of these techniques have been demonstrated in this article and shown to achieve performance improvements anywhere in the range of 10% to 1700%. Fortunately, the same techniques also seem to benefit the other CPU architectures. This is good news since it means that usually one optimized implementation will perform well across a broad range of CPUs.

The biggest hurdle to developing high-performance applications under Linux is the current lack of sophisticated performance analysis tools. The relative lack of such tools is not surprising; while most commercial Unix vendors have tools for their own architecture, few, if any, are multi-platform. To some degree this is inherent in the problem, but there is no question it would not be very difficult to create even better portable performance-analysis tools.

Linux is what makes low-cost Alpha-based Unix workstations a reality. While Digital UNIX currently comes with better compilers, runtime libraries and more tools for the Alpha, the price difference is such that one can easily make up for the performance difference by spending a little more money on a faster

machine. Also, development of gcc and better libraries doesn't stand still. However, since most work is done on a voluntary basis, it does take some time. Even so, Linux is already a highly competitive platform for integer-intensive applications. For floating-point intensive and especially FORTRAN applications, things are not yet so mature. Fortunately, if one cannot afford to wait for a better compiler, there is always the option of purchasing one of the commercial FORTRAN compilers available for Linux/Alpha.

Acknowledgments

The author would like to thank Richard Henderson of Texas A&M University and Erik Troan of Red Hat Software for reviewing this paper on short notice. Their feedback greatly improved its quality. Errors and omissions are the sole responsibility of the author. This article was first given as a speech at Linux Expo 97 on April 5.

References



David is a graduate student in the Ph.D. program of the Computer Science department at the University of Arizona. He plans on graduating in August 1997 and to finally get a "Real Job". He was involved in the Linux/Alpha port and has been sticking around in the free software community ever since. When not playing with computers, he enjoys the outdoors with his lovely wife. He can be reached via e-mail at David.Mosberger@acm.org.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)


Mistaken Identities

Marjorie Richardson

Issue #43, November 1997

Photo mixup.

In the September 1997 issue we printed author pictures for the “Linux for Embedded Systems” article, when indeed we had not received pictures for either Sandor Markon or Kenji Sasaki.

	This picture is actually Eric Harlow, author of “Building an ISP Using Linux and an Intranet”.
	This picture is actually Jan Rooijackers, author of “Quota: Managing Your Disk Space”. Jan got his picture printed twice—lucky guy.



[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Metro-X 3.1.5

Mark Nassal

Issue #43, November 1997

Metro-X is extremely robust and easy to configure.



- Manufacturer: Metro Link Inc.
- E-mail: sales@metrolink.com
- URL: <http://www.metrolink.com/>
- Price: \$99US (Linux)
- Reviewer: Mark Nassal

One of the best things about UNIX is that it allows you to customize and configure almost everything. This is also its greatest downfall. Nine times out of ten, configuration requires editing long and sometimes cryptic text files. One of the best examples of this phenomenon is setting up an X server.

Most changes are made on a trial-and-error basis. This method entails opening, editing, saving and trying the configuration file over and over. When it finally works the way you wish, you pray you don't have to change it again. Of course, as soon as you upgrade your graphics card, it's editing time again. You can forget about changing color depth on the fly—it's too bad if some of your applications work only at 256 color and the rest allow true color. If you want to use those 256 color applications, you will have to run all your programs at 256 colors.

The X configuration is something Linux users have had to endure like an ancient tribal ritual. It is often accompanied by screams of frustration, loss of sleep and hardware flying across the room.

Well, all that has changed with the advent of Metro-X. Metro-X is an X server by Metro Link Incorporated. It is extremely robust and easy to configure. Metro is available for most of the leading x86 Unix platforms (Linux, Solaris x86, System V 4.0, UnixWare and SCO). Red Hat has also started shipping Metro-X with their commercial 4.x distribution of Linux.

Configuration is a snap. You are guided through the process by a clean, well thought out Motif based GUI (see Figure 1). If you make a mistake or upgrade hardware, changes can be made in a matter of seconds.

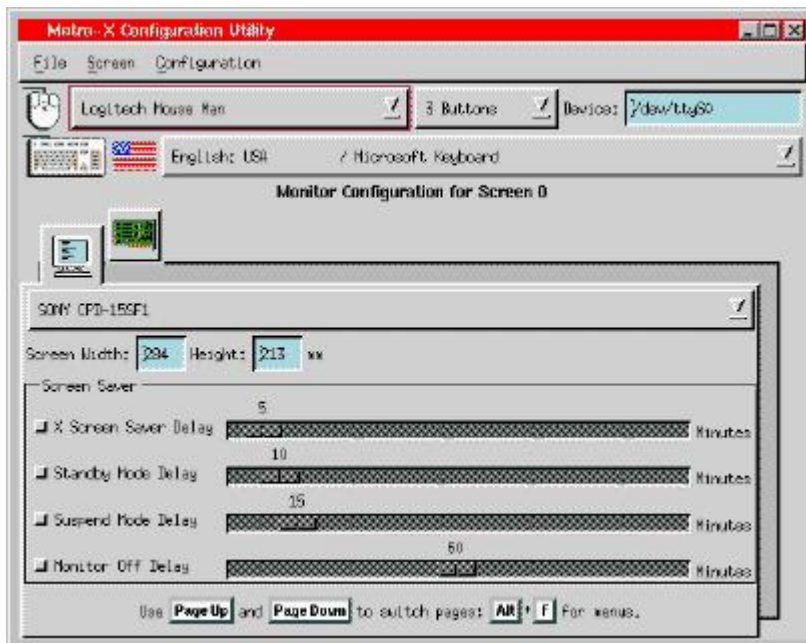


Figure 1. Configuration GUI

System Requirements

Metro Link recommends a minimum configuration of 8MB of RAM and 12MB of hard disk space. I would recommend a minimum of 16MB of memory and a 486 33MHz CPU, as less than 16MB doesn't leave much room for applications. Note that XFree86 must be on the system before Metro-X can be installed, since Metro-X uses libraries and fonts installed by XFree—Metro-X is not a complete replacement for XFree86, just for the XFree86 X Server.

Price

Pricing is very reasonable. A single-user license is included with the Red Hat Linux 4.x boxed set which sells for \$49 US. A general Linux version is \$99 US, and all other platforms list for \$199 US. Metro-Link also offers Motif, OpenGL and Metro-Media at an additional cost.

Installation

If you are putting the product on a Red Hat Linux system, installation is easy using the Red Hat Package Manager (RPM). Log in as root and enter:

```
rpm -i /path/to/Metroess.3.1.5-2.i386.rpm
```

To ensure that Linux uses Metro-X instead of the default XFree86 server, the X11 link must be changed. Create the new link using the command:

```
ln -sf /usr/X11R6/bin/Xmetro /etc/X11/X
```

Note that some systems put this link in `/usr/X11R6/bin`.

Configuration is done through the Metro-X GUI (see Figure 1):

```
/usr/bin/X11/configX
```

X11 will automatically load with the GUI as the foreground process.

The first drop-down box allows the user to select from fourteen standard input devices. There is support for bus mice, PS2 mice, keyboard mice, track balls and several touch screens. And, of course, it allows you to choose the button configuration, e.g., 1, 2, 3 or three button emulation.

The next drop-down box is for keyboard configuration. The keyboard list is pretty impressive. There is support for most languages and for my favorite, the Microsoft keyboard. Since I spend so many hours at the computer, it's nice to have an ergonomic model. I was surprised to see my Microsoft keyboard listed—it's a non-standard item for a Unix system.

Table: Supported Video Cards

Next is the video card configuration. Metro supports thirty six brands of video cards (see Table 1). For specific card information visit the Metro Link web site at <http://www.metrolink.com/products/metrox/cardlist.table.html>. They have created a detailed table which lists supported color depths and resolutions for each card. Most of the Diamond, Number Nine and Video7 cards are listed. The support for Trident is a little weak; only one of their cards made it to the list (Trident 8900).

Color depth, frequency, virtual resolution and physical resolution are set by the click of a button. Depending on the card, color depths from 16 to 16M can be chosen, as well as virtual and physical resolution range from 640 x 480 to an impressive 1600 x 1200. The nice thing about the GUI configuration is you can quickly change color depth and resolution depending on your application

needs. I have several programs, such as Applixware 4.2, that don't work right above an 8 bit pallet. However, I like to do all my graphics editing in true color. With Metro-X, I can change the video configuration to match my applications in a matter of seconds.

The **Monitor** button displays a huge list of common monitor types, and automatically creates the allowable frequency range and synchronization. If your monitor isn't on the list, don't worry—there are lots of generic monitor types to choose from. You can even enter the exact screen size and select the screensaver delay time. If you have a energy star monitor, you can set standby, suspend and monitor off.

When everything is set up the way you want it, save the configuration file and exit.

Log in as a regular user and start the X server:

```
/usr/X11R6/bin/startx
```

If you need to change the color depth or resolution, simply rerun **configX** while logged in as **root**. Make the necessary adjustments then log back in to X Windows.

Enhancements

One of the more intriguing features of Metro-X is multi-screen support. If you use Metro-X MGA video cards, four monitors can be operated simultaneously (see Figure 2). Applications can be loaded into their own screens and controlled by a single mouse and keyboard. A GUI based utility is provided to handle the physical screen layout and mouse movement (see Figure 3). I would love to play around with this configuration, but my wallet won't cooperate.

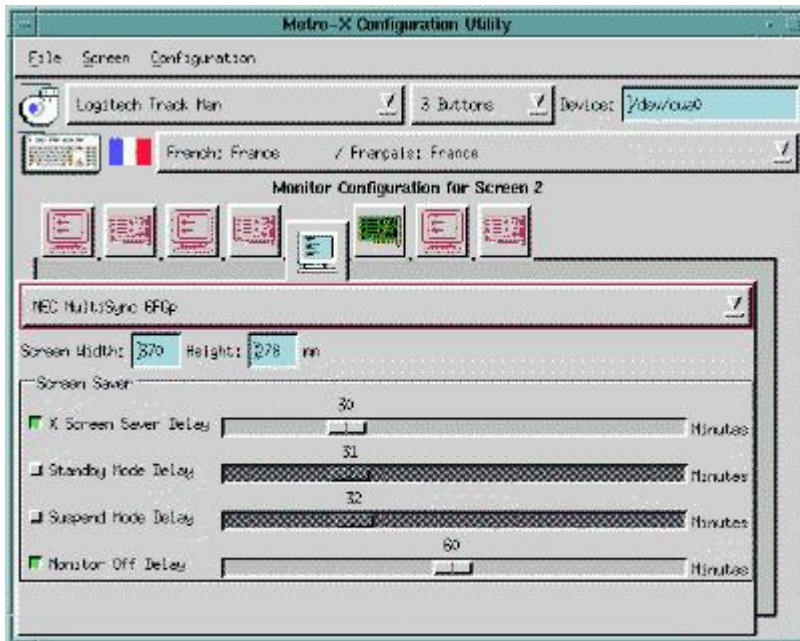


Figure 2. Monitor Configuration Screen

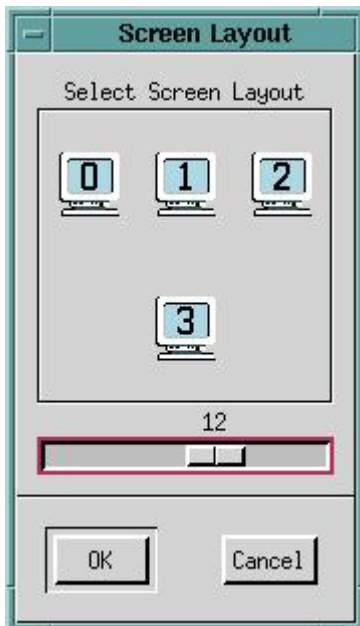


Figure 3. Screen Layout GUI

As with most modern X servers, multiple screens and virtual resolution are supported. I cringe whenever I have to work on a system that doesn't allow these features. I normally work in several applications at once, and trying to cram multiple windows into a single screen is a nightmare.

There is also the fail-safe escape method. If the X server freezes or your mouse becomes disabled, Metro-X can be brought down by pressing the ctrl, alt and backspace keys simultaneously. This is a lot quicker than pressing ctrl-alt-f1 to get a virtual console, then killing the X Server process.

If you're into video capture, MPEG playback or video conferencing, Metro Link sells a video suite called Metro-Media. The application runs on Metro-X with Motif and can be used by developers and end users.

For developers and end users who use Motif-based applications, Metro Link also supplies Motif 2.0 at \$199US.

Performance

I immediately noticed a performance increase when Metro-X was installed on my system. It seems to be a very stable environment. Several programs that misbehaved under XFree ran great with Metro-X. Before Metro-X, Netscape was continually giving me trouble—now, it runs like champ.

Support

Support options are dependent on where you purchased the package. If you received it from Red Hat, Red Hat provides the support. Red Hat offers on-line support at <http://www.redhat.com/> and phone support at 1-800-546-7274.

If the purchase was made directly from Metro Link, they provide support through e-mail at tech@metrolink.com and phone support at 1-954-938-0283.

I have yet to encounter a problem that required a tech request, so I can't speak to the quality of support.

Overall impression

With the ease of configuration, stability, available options and enhancements, I think Metro-X is a real bargain. I recommend it highly to anyone who wants to spend his time producing instead of configuring.



Mark Nassal is a Unix Systems Engineer for Harte Hanks Data Technologies in Billerica, Massachusetts. His areas of concentration are Intra/Internet site development and Systems integration. Primary administration is performed on SunOS, Solaris and Linux systems. For relaxation he enjoys photography,

graphic arts and thrash music. Mark can be contacted at nassalm@tiac.net or <http://www.tiac.net/users/nassalm/>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

SCO OpenServer

Ken Collins

Issue #43, November 1997

OpenServer may be free, but it's not "open".



- Manufacturer: The Santa Cruz Operation
- Phone: 1-800-SCO-UNIX
- URL: <http://www.sco.com/>
- Price: \$19.00 US (media)
- Reviewer: Ken Collins

Compulsive operating system collectors can now add another partition to their hard disk and another OS to their stockpile. The Santa Cruz Operation (SCO, <http://www.sco.com/>) has a new strategy for attracting users to its OpenServer Desktop System. SCO is giving it away for free, or at least nearly for free.

Though the company could claim 40% of the Unix market share a couple of years ago, it's now facing serious pressure from Windows NT and of course, Linux. Because the OS is priced similarly to NT, it has traditionally been out of the reach of students, educators and the idle curious. While there are a vast number of commercial applications running on OpenServer (it's often employed as a base for retail database systems, thus its market share), the company obviously wants to interest a new generation of developers and administrators.

To further its aims, SCO has made its single-user license free. The catch is that you still have to purchase the media. The OpenServer package, which includes a CD-ROM, a boot disk, a disk of drivers and a small pamphlet costs \$19.00 US. If you're used to downloading your distributions at no cost, that's not an option, but SCO has also been handing out their OS gratis at trade shows. You won't be able to download the source anywhere. OpenServer may be free, but it's not "open". Additionally, the single-user license is intended for educational use only. If you want to run OpenServer as part of your business, you're supposed to purchase a license.

Getting started

While you're at the SCO site signing up for your license (<http://www.sco.com/offers/index.html>), you might also want to take a look at the on-line documentation for the installation procedure (<http://www3.sco.com/Products/freeunix/suppinst.html>) and the hardware compatibility handbook (http://scaffold.sco.com/chwp/owa/hch_search_form). Unfortunately, the pamphlet included with the installation disks only includes the following instructions for first-time users: "Boot from the supplied boot floppy and follow onscreen instructions."

While not overly difficult, the installation is far from intuitive. In fact, the first screen will throw all but the most adventurous installers. It asks for your installation media and offers a number of options, but gives no indication that you can install from an IDE CD-ROM. If you use the SCSI CD-ROM setting, follow the on-line documentation directions for the other settings and if the drivers are on the disk, it should recognize your drive. The rest of the process is familiar to Linux users. You can select portions of the software to install (including a lot of support for Novell networking), and you can use **fdisk** to partition your drive.

While you're going through this process, OpenServer is merrily overwriting your master boot record and wiping it free of LILO. After some serious wrangling, I gave up on getting the OS to cooperate with Linux. With LILO installed in my Linux partition and OpenServer in its own half of the drive, I couldn't find a boot loader that would get SCO to start up, nor could I coax OpenServer into recognizing my Linux partition from its boot prompt. For the time being, I let OpenServer have its way with my system, fdisking back and forth when I needed one or the other, but serious dual users will have to come to terms with this problem. It may be a job for V Communications' System Commander.

Inside the OS

Once inside the OpenServer environment, you're given the opportunity to log on as root or proceed to an SCO-centric xdm login prompt. If you're wary of X,

the system offers as many virtual consoles as you have function keys. However, without bash, tcsh or whatever shell you favor, the console is pretty tedious. OpenServer comes with the Korn shell, the C shell and its own **scosh**. This means if you're a filename completion junkie or you're fond of browsing your command history with the arrow keys, one of the first things you'll want to do is port the appropriate shell to your system.

The scosh is a world unto itself and deserves some mention. The name seems to suggest that it's a shell, but it's closer to an MVS era menu system. After invoking scosh, it takes over your console or xterm and pops up a calendar, a series of menus and a listing of the current directory. The xterm becomes mouse sensitive, and you can use it or the cursor keys to select from the various options. It presents a fairly complex interface; it provides a front end to navigate the file system, check e-mail, edit files, set permissions, print, create archives and move files around. Even so, SCO's addition to the world's shell archives isn't much of a shell. It does everything it can to keep you away from Unix, and devotees of the elegance and ease of the command line may find scosh an unpleasant and unnecessary feature.

The Desktop

In any case, with the OpenServer desktop up and running, scosh's text interface is rendered moot. Most of its functionality is right on the desktop, which combines features of the Macintosh and Windows GUI. You can browse the file system Mac style, with windows, files and folders, but copying documents requires a click of the right mouse button as in Windows 95. There are also the ubiquitous "File" and "Edit" menus at the top of every window. Lest you forget you're using Unix, the window manager features a four-panel desktop panner, and you can bring up a shell from the Unix icon. Pop-up menus that follow the mouse mimic most of the menu system functions and round out the desktop's basic features. Anyone familiar with FVWM will feel right at home with the look and feel of SCO's window manager, and it maintains a solid balance between functionality and desktop clutter.

You'll probably first want to explore the system administration folder, which offers some graphic front ends to the process of fine tuning SCO. I say some because often the interface reverts from a full, icon-based system to a window with a text prompt. Unfortunately, the treatment is pretty uneven; the file-system manager presents friendly disk pictograms that you can mount and configure while the floppy-file-system manager leaves you grappling with its multiple-choice text menus.

While noodling around in the administration folder, I found the hardware/kernel manager pleasantly easy to use—it was pretty simple to build DOS file-system recognition into the kernel when I wanted to get at a floppy. However,

after the joys of RedHat's configX, SCO's video configuration manager was somewhat disappointing. It doesn't offer much flexibility when it comes to fine tuning your setup; you can select your monitor, video card and whatever resolutions are supported. If you get the wrong one and your screen is suddenly tweaked, it's not easy to back out to a better choice. My video card wasn't included in the defaults, so I was sent scrambling for SCO's advanced hardware supplement, which is included on the CD-ROM.

Within the networks folder, similar surprises awaited me. I was hoping to set up PPP, but there were options only for configuring a LAN or a WAN. As a last act of desperation, I consulted the OpenServer help system. To SCO's credit, the help application is well constructed and easy to use. Though it's not innovative—it uses a web browser style interface—it's easy to navigate, and help menus in every window provide the opportunity for context-sensitive clues. The documentation is no worse than your average HOWTO, though they lack the personal touch of Linux collaborations. I was pleased to find numerous examples in both the help system, and, after abandoning the GUI, in the configuration file comments. (It turns out that PPP can be configured partially as a WAN network and partially using the UUCP dial-up tools.)

Beyond noodling with the system configuration, there isn't much else to do with OpenServer. There are tools to define aspects of the display, and the distribution includes the usual generic calendar application, text editor and even Mosaic.

Unfortunately, while SCO provides an OS, it doesn't offer much in the way of a distribution. And, before you're able to port your favorite applications, you may have dig up a compiler. After several tries, I finally got the installer to recognize my free license for the OpenServer Development System. I had to install the non-developer version, then use the SCO software installer to import the developer version from the CD-ROM. This process provides a prompt for the developer license, and this time, the free license worked. It never did accept the license from the main install screen. This problem should be partially remedied by UnixWare, which SCO has just started offering for "free" (again, \$19.00US). UnixWare includes Netscape Navigator Gold, the UnixWare Software Development Kit and Netscape's FastTrack Server, so it will save you downloading time and give you more to play with.

Most of what you gain by installing SCO is an appreciation for Linux and its model for system development. Though many of my complaints about OpenServer could easily be turned against various Linux distributions, OpenServer clearly demonstrates that Linux is at least comparable to and may even surpass SCO's commercial Unix in its general functionality.

I can't say that I found a truly compelling reason to use SCO, nor did I uncover any features that Linux seems to lack. Moreover, if I was forced to use OpenServer, I'd have to spend a considerable amount of time outfitting it with the standard shells and tools that I've come to expect with any Unix. Given the OS's restrictive licensing agreement, it's difficult to believe that SCO will be able to attract any serious new believers even at their clearance price. But, if you can pick OpenServer up for free, it's the best proof that Linux has really become a commercial-grade product.



Ken Collins is an Internet software developer at Neoglyphics in Chicago. To keep his right brain from crystallizing he tries to stay current on Soviet history, dabbles in postmodern theory and writes for the New Art Examiner. He can be reached at panic@suba.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Lifebook 420D Notebook Computer

Michael Scott Shappe

Issue #43, November 1997

It is neither fancy nor overly innovative in its design (interior or exterior), but so far it appears to be a solid, stable machine, difficult to beat at its price point.



- Manufacturer: Fujitsu PC Corporation
- Phone: 800 8FUJITSU
- URL: <http://www.fujitsu-pc.com/>
- Price: \$1599.99 US
- Reviewer: Michael Scott Shappe

The Fujitsu Lifebook 420D is the newest and least expensive addition to the Lifebook line of notebook computers. It is neither fancy nor overly innovative in its design (interior or exterior), but so far it appears to be a solid, stable machine, difficult to beat at its price point. Perhaps more importantly, the Lifebook makes a good, inexpensive Linux laptop machine, with a couple of comments.

A Compact, Not a Sports Car

The Lifebook reminds me strongly of my first new car, a Pontiac Sunbird. It has a solid feel to it, with keys, latches and components that click appropriately when used. Its outward styling is not very fancy, looking pretty much like most

of the other models in its class. For its size, it's a little heavy, at 7.3 pounds. It's not capable of going extremely fast, sporting a Pentium 120 as its CPU, non-MMX processor. It has adequate headroom and leg room, but not lots of it, at 16MB of RAM, expandable to 72 (The model is actually advertised as having only 8MB of RAM, but current models are shipping with a "free upgrade" to 16. My guess is that Fujitsu realized at the last minute that supplying mere 8MB was simply not adequate.) and a 1.08GB hard drive. It does not get superb mileage, having a nickel-metal-hydride (NiMH) battery of only two hours duration, although there is an advertised option for a lithium-ion battery.

On the other hand, for a relatively low sticker price, it comes equipped with many of the things people have come to expect in a personal computer today. Fujitsu preloads Windows 95 (fear not, dear reader, I'll fix that problem shortly), Microsoft Works, Intuit's Quicken PE and CyberMedia's FirstAid suite. In a moment of lucidity, Fujitsu decided to include both the Windows 95 and the bundled software on CD, so there's no need to make backup floppies just to have a pristine copy of the bundle. There's a floppy drive and a 10x CD-ROM, although as shipped they share a single multi-function bay; an optional cable allows you to hook up the floppy drive externally.

Fujitsu gave the Lifebook the usual array of ports: serial, parallel, SVGA, an IrDA-compliant infrared transceiver, a port-replicator dock, headphones out, microphone in and two PCMCIA slots in the typical two-of-Type-II-one-of-Type-III configuration. The screen is a 11.3 " Dual Scan" that is fairly readable. Directly underneath the screen is a status panel, with indicators for power, suspend-state, charging state and disk access. The "suspend" button is right next to this panel, large enough to be obvious, but out of the way enough to be impossible to hit by accident. This means that you can not put the machine to sleep or wake it up unintentionally—a definite plus.

The keyboard has about 80 keys, including almost all the typical PC-keyboard keys and the more recent Start and Menu keys. Finally, it sports two speakers (one on each side of the status panel, right beneath the screen's hinges), a condenser-microphone hidden on the left side near the front of the unit, and SoundBlaster-compatible audio hardware.

Great. What about Linux?

The most difficult part of getting Red Hat Linux 4.1 up and running on the Lifebook 420D was getting it physically installed. But as an experienced user, even that wasn't a big problem. Quite honestly I made it a little harder on myself than it had to be by making some of the specific choices I did.

For one thing, I wanted to keep a "minority" Windows 95 partition, primarily for playing games. I bought the machine primarily as a writing utensil (Emacs may

well be as bloated as MS Word, but at least it's free and doesn't crash), but I do like my diversions. Most of the games I'm interested in aren't even out for the Macintosh, let alone Linux.

For another thing, I acquired neither a modem nor the optional external floppy adapter. In the first case, I decided against it because my wife and I already own a modem (although I didn't acquire a serial cable at first, either), and I wanted to wait out the modem speed wars before buying another. In the second place, I decided that I simply would not require both the CD-ROM and the floppy drive at the same time all that often, if at all, which so far has proven correct.

The result was that, to accomplish all my goals, I had to rely on the ability to swap the CD-ROM and floppy drive while the machine is suspended. First, I cleaned out all the bits of bundled software I was fairly sure I would never, ever use, which included MS Works, Internet Explorer and a number of other odds and ends. I then copied Red Hat's FIPS utility from the CD to the hard drive, suspended the machine and swapped the CD-ROM out for the floppy drive. I woke the machine up, built a simple DOS boot floppy and copied FIPS over to it.

Before rebooting, I ran the bundled disk defragmenter to make sure everything was well "packed down". This done, I rebooted using the floppy disk, and used FIPS to non-destructively shrink the Windows partition to about 400MB, leaving 600 free for Linux (which so far has proven adequate). I then shut down, swapped back to the CD-ROM, rebooted from the hard drive into Windows, and ran Red Hat's "autoboot.bat" to boot off the CD. (Red Hat could have made this even easier by making the CD directly bootable; the 420D's BIOS can boot from CD-ROM.)

From here, the installation was as easy as any Unix or Unix-like install I've ever done, and I've done quite a few, including Ultrix, SunOS 4.1 and Solaris 2.5. I used Linux FDISK to set up five partitions, one primary: root (50MB); and one extended with three logical: /usr (300MB), /home (230MB) and swap (16MB). I installed most of the typical packages, including TeX and XFree86, although I later wound up pruning some of the documentation and games to gain back extra space. In the end, however, I was able to load what I wanted and needed and still have room to spare.

In retrospect, since the /home partition is going to be entirely for me, and primarily for storing text files, it could have been smaller, and /usr a bit larger (perhaps 350MB for /usr and 180MB for /home). As it is, I wound up migrating X11R6 (44MB) over to /home and putting a symbolic link back to /usr/X11R6, which works fine and balances out the disk space adequately. The result leaves me plenty of room for the Linux kernel sources and with room to grow.

Once in place, everything more or less worked with one glaring exception, which I'll get to in a moment. The track pad appears as a PS/2-style mouse (and is touch-sensitive so you can tap instead of click the left-button), the keyboard as a standard PC-101 keyboard and the screen appears as a typical VGA screen. The Lifebook's BIOS is APM (Advanced Power Management) compliant, so the kernel's APM extensions are able to work perfectly (although Red Hat 4.1 did not include the APM utilities). I even tried suspending the machine, using the suspend button, while in the middle of compiling a program; the system resumed when I told it to without any problem.

All the standard ports are recognized, and the PC Card slots appear to work fine with the pcmcia-cs drivers. (Red Hat 4.1 shipped with the pcmcia-cs 2.8.25 drivers and the 2.0.27 kernel, which may well work adequately; I had upgraded to 2.9.5 and kernel 2.0.30 before testing the PC Card slots, however.)

Minor Quibbles

I do have a few minor quibbles when comparing this unit to what might be my more "ideal" machine. On the other hand, I recognize that most of what would make up my "ideal" machine would result in a price that was well outside of my range. Looked at in the larger picture, the trade-off of features for availability represented by the Lifebook 420D is an excellent one.

The Dual Scan screen is fairly good as such things go, but nothing quite compares to Active Matrix. Even this DS screen suffers from ghosting and a little bit of cursor "submarining". However, it is fairly fast and pretty crisp. Once I got X-Windows to run correctly (more shortly), I was greeted with surprisingly sharp color. Given that Active Matrix would probably add \$1000US to the price tag, I'd say that this particular trade-off is well worth it.

The lack of either a built in SCSI port or a modem is a bit of a bother. A SCSI port, in particular, would be useful for hooking up my Jaz drive so that I could do backups (1.08GB hard drive, 1GB Jaz platter). The modem's absence is less important to me, personally, but seems very strange in this age of Internet madness, particularly since they **do** bundle various on-line service programs by default. The inclusion of these two things, however, would probably have added \$300-\$400US to the sticker, and since I do already have a modem, and PC Card SCSI adapters can be had for just over \$100US, this trade-off, too, is one I think I agree with.

The Lifebook comes with a "port replicator" docking port, but the existing port replicator product for the Lifebook 500 series does not work with it, and Fujitsu has yet to release the "mini-docking station" that will. They have not even announced what the docking station will include, or at least, if they have, I haven't seen it anywhere on their web site. The annoying part about this is that

the Lifebook 500 series replicator sells for \$189US, but the docking station will, according to a salesman at Computer City, sell for \$369US. For that price, the docking station had better have the two things I just griped about: a SCSI port and a modem.

The Big Quibble

Everything I've described so far falls under the heading of acceptable trade-offs in a budget notebook computer. My largest complaint does not. Fujitsu has made one particular design decision which, despite all the great things about this unit, makes it very difficult for me to give an unalloyed endorsement to this product.

The Lifebook 420D uses a NeoMagic 128v model 2093 graphics chip to drive its LCD panel and SVGA-out port. This has become a very popular choice with a number of portable manufacturers, including Dell, Toshiba, IBM and DEC. From what I can gather, this decision has been taken in part because NeoMagic has succeeded in squeezing the entire video works, including hardware acceleration, onto a single chip, rather than a set of chips, a major win for price, packaging and power consumption, no doubt.

Looked at without concern for specific operating system support, this is a perfectly good trade-off. The NeoMagic chip produces excellent results, and I have little doubt that part of the reason the Dual Scan screen is so usable is that the NeoMagic chip is so good at what it does.

Unfortunately, according to folks at the XFree86 project, NeoMagic has chosen to keep the programming specifications of their chip proprietary. The result is free software authors such as the XFree86 group cannot obtain the specifications to write and release drivers that will take full advantage of the chip without compromising on the principle of free software. This seems to be a very backward policy on NeoMagic's part given that larger competitors such as Diamond and Matrox have begun working more openly with the free software community.

What this means for the end user is that they're left with two not-so-great choices. The first is to run XFree86's VGA16 server, which has a good enough "generic" driver to run even the NeoMagic chip at 800x600, but in only 16 colors. The second is to run Xi Graphic's Accelerated X LX product, which does have a complete driver for the NeoMagic chip, but costs \$200US—twice the cost of the more common AX desktop product. MetroLink has informed me that they are considering supporting the NeoMagic chip in the near future, so you may eventually be able to use Metro-X as a commercial alternative, as well.

With either current solution, a “trick” is required either in the kernel itself or in lilo.conf to force the video system into “graphics” mode immediately at boot-up. Without this trick, the 800x600 image shows up offset by an inch to the right, leaving a black gap on the left and cutting off the rightmost portion of the desktop entirely. Unfortunately, using this solution means that you lose the ability to track boot-time messages as they happen. This isn't a big problem for me; I've used Macintoshes, which have no boot-time messages, for years. I've also left an alternate, text-mode configuration in my lilo.conf for emergencies. It also means you have to use the X Display Manager to present a login screen, but most current Linux distributions, including Red Hat, provide an easy means of doing so in the form of a special “run level” in /etc/inittab.

The details of how to configure these programs and lilo.conf are beyond the scope of a review. If you want more information, take a look at <http://www.publiccom.com/web/mikey/lifebooklinux.html>, a page I've set up specifically to cover these issues.

Accelerated X is a good product. However, I prefer free software. The whole point of my buying this particular model was that I didn't want to shell out a lot of money, so spending another \$200US after purchase doesn't thrill me. Fortunately, for my own uses, 16 colors is plenty, so for now, I'm content with using the XF86_VGA16 free server.

The Bottom Line

The Fujitsu Lifebook 420D is a solid, dependable (so far), and above all affordable notebook computer. While not endowed with all the latest gadgetry or the fastest hardware, it has more than enough power to run Linux well, in large part because Linux takes such good and efficient advantage of the hardware. While not designed for very high speed or for very great range, it's an excellent and inexpensive selection.

The only thing that prevents me from giving this product a complete endorsement is the NeoMagic chip issue. I have politely expressed my opinion of their proprietary attitude to both Fujitsu and NeoMagic. I can only hope that some heed of the Linux market is taken by either or, preferably, both companies.

If this particular issue *doesn't* bother you or if, like me, you're willing to work around these problems if it means you can have a portable computer without taking out a second mortgage, then by all means check out this product. If, on the other hand, you decide against the Fujitsu 420D specifically because of the NeoMagic issue, then I suggest you express that decision in a *politely worded* letter to both Fujitsu and NeoMagic. It's a market-driven world; the only chance

we have of getting respect is to demonstrate that we are a market worth considering.



Michael Scott Shappe is a senior programmer for AetherWorks Corporation, a technology startup in St. Paul, MN. He's spent the last 10 years hammering on Unix systems of various stripes and addicting the unsuspecting to the Internet. You can peruse his personal web at <http://www.publiccom.com/web/mikey/> or send him e-mail at mshapp19@idt.net.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

AcceleratedX CDE and Display Server for PC Unix

Bradley Willson

Issue #43, November 1997

The wide variety of supported hardware makes this product a better choice in high-demand environments and with aging equipment.

- Manufacturer: Xi Graphics, Inc.
- URL: <http://www.xig.com/>
- E-mail: sales@xig.com
- Price: AcceleratedX Display Server \$99.95US AcceleratedX CDE with Display Server \$274.95US
- Reviewer: Bradley Willson

The "X inside inside X" logo flashed on the screen then quickly went away. What happened next gave me cause for celebration. My old NEC Multisync II flashed graphics faster than ever before. My 9FX Vision 330 card was among more than 449 other video cards that AcceleratedX reportedly supports. I also found a "closer" match to my ancient monitor's specs among the 100 plus list of supported monitors and generic settings. I had more control over colors and display than ever before. While talking to pre-sales tech support, I found there are more features available that my hardware does not support. High-end equipment can support the 8/24 bit display and gamma correction features.

Figure 1. CDE Toolbar

Before testing the new server, I loaded a copy of **xbench** and ran it on my stock configuration of the X312_S3 server and FVWM95-2. I used the xbench default parameters that run each test three times for ten seconds per run. I performed the same test on AcceleratedX, then compared the results. Overall process times averaged thirty percent faster with the AcceleratedX server.

Next, I loaded the AcceleratedX Common Desktop Environment. My initial euphoria faded at that point, and I began to pack up the pointy hats and party

favors. The act of installing CDE had clobbered the display server installation, and there was nothing in the documentation that warned me otherwise. A quick e-mail conversation with tech support confirmed the importance of sequence and included an assurance that they would add an addendum to their release notes to warn users of this requirement. I reinstalled the display server and resumed testing.

Further investigation revealed that the X Windows configuration was also modified. The X386 directory was copied to X386.old and then re-populated per CDE specifications. Other directories like fvwm95-2 were removed from the parent directory altogether. Another annoyance came from the way CDE bypasses loading /etc/profile. None of my scripts worked. /sbin, /usr/local/bin and /usr/local/sbin were no longer in the path. After making some symbolic links to the missing directories and editing path statements, the system was back up and running, mostly as it was before. A few more words need to be added to the release notes to help customers configure their systems for test-driving CDE.

Figure 2. Application Builder

Hindsight is a great tool, if only it could be applied in reverse. I wanted to retain my configuration while testing the new software. The philosophy behind CDE does not take that scenario into account. It is fully intended to be the primary X window manager on the system, with **twm** as the failsafe. When the opportunity presents itself again, I will load these packages after the kernel and before adding any other applications. That way the configuration will remain consistent throughout.

Description

The CDE's front panel is impressive at first sight. Ten frequently used programs and utilities are placed for first priority use. Other, less-used programs are embedded within pop-up menus. The workspace manager is placed front and center, configured with four unique workspaces. An "LED" flashes to indicate a process is in work. A padlock icon launches a session display lock. The panel can be positioned anywhere on the desktop and be minimized or maximized with buttons on the panel ends. The package includes several backgrounds, screen-savers and desktop color schemes, making it easy to customize each workspace according to individual taste.

Figure 3. Style Manager

The CDE features several useful applications, including a multi-user planning calendar with drag and drop appointment scheduling and to-do items, a file manager similar to **xfm**, a simple text editor with spell checking, a MIME

capable mail handler, a graphic printer manager, an application manager with numerous other utilities and the quintessential trash bin. There is even a Motif application builder for the programmer types. The multi-function calculator serves the financial, programmer and scientific users with the click of a button. Even the xterm is configurable on the fly.

To customize a control, one simply clicks on the icon with the right mouse button and then selects "Add Sub-panel" from the pop-up menu. The sub-panel is then ready to accept new application and utility items. Sub-panels and applications are removed in the same manner. The only drawback is additional controls have to be added by the system administrator by editing text files.

Installation

Figure 4. Help Viewer

The installation process for both packages is simple and reasonably straightforward. The CDE installation consists of mounting the CD and running **dtinstall**. The script takes care of the file copy and administrative tasks in one step. The display server installation is explained in the user's manual. The process entailed changing to a specific directory, decompressing the package from two floppy disks and running two scripts.

Technical Support

Xi Graphics' technical support can be summed up in two words, quick and effective. Send your inquiry to support@xinside.com or support@xig.com. If you just need information, send a note to info@xig.com. You will receive an automated reply with a generous helping of information texts and FAQs that are available from their mail-server. You can also call their pre-sales tech support and sales line at 800-946-7433, full-service tech support at 303-298-7478 or fax your questions to 303-298-1406.

Figure 5. Printer Window

Comparison with other products

This is the first commercial X server I have had experience with, since I normally take the "free" route. When compared to XFree86, it shows definite signs of improvement. The wide variety of supported hardware makes this product a better choice in high-demand environments and with aging equipment. I found CDE to be easier to set up and administer in a networked environment than XFree86. The robust selection of built-in applications add value to any system. The printed manuals are laid out in a logical fashion, making it easy to find most of the answers. The on-line help is more graphical than **xman**, but the

information is focused on helping the user, not the system administrator. In general, I liked AcceleratedX and the Common Desktop Environment. I certainly recommend it for mission-critical commercial applications and for the "serious" home user.

Company Profile

Xi Graphics, Inc. is located in Denver, Colorado, and employs a staff of around 20 talented individuals. Their web-site can be browsed at both <http://www.xinside.com/> and <http://www.xig.com/>. XiG.com is the preferred address as they have recently changed their company name from X Inside to Xi Graphics.

Available

Xi Graphics has a widely dispersed list of authorized resellers, located in Canada, USA, Europe, the Middle East, Africa, Australia and Japan. Either browse their web-site or send e-mail to sales@xig.com to find the dealer nearest you.

Installed base

US West in Minnesota, University of Minnesota, University of Texas Austin and University of North Carolina are among notable users of AcceleratedX and the Common Desktop Environment.

Platforms

If it's Intel, has Linux or FreeBSD and X11R6 installed, these products will run on it.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

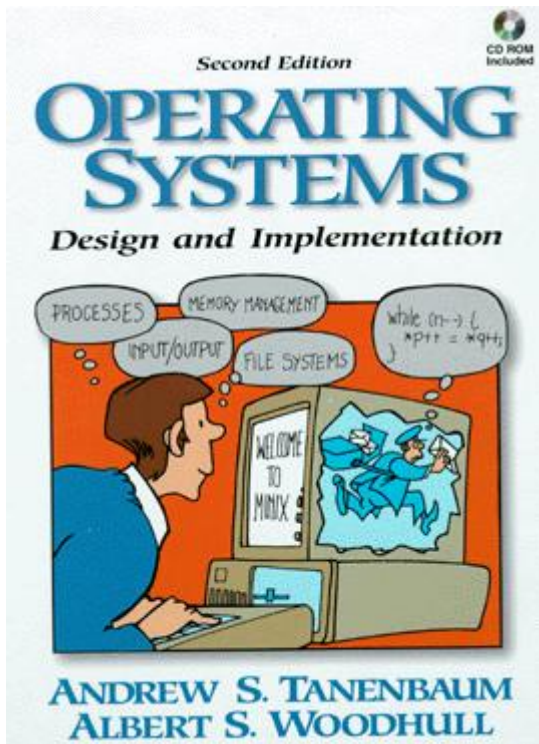
Advanced search

Operating Systems, Second Edition

Boycho Peytchev

Issue #43, November 1997

When reading this book, a knowledge of both C and assembly language is helpful but not required.



- Authors: Andrew S. Tanenbaum and Albert S. Woodhull
- Publisher: Prentice Hall
- <http://www.prenhall.com/>
- Price: \$66 US
- ISBN: 0-13-638677-6
- Reviewer: Boycho Peytchev

Operating Systems: Design and Implementation is an introductory book designed for people who have already dealt with the basics of programming. When reading this book, a knowledge of both C and assembly language is helpful but not required. In particular, not knowing assembly will not prevent you from understanding the book, but it may slow you down.

The first chapter of the book is an overall introduction to the topic of operating systems. It starts with an explanation of what an OS is and a short historical summary on the evolution of operating systems. Then, the basic concepts of operating systems are explained and the different possible structures for an OS are introduced. The introduction ends with an overview of the remaining chapters of the book.

The style of the authors is very straightforward, making the information easy to follow and understand. The text doesn't get side-tracked into irrelevant topics, a problem found in many text books.

Another strong point of this book is that it is not only a theoretical introduction to operating systems programming but a practical one as well. The included CD-ROM contains MINIX 2.0 complete with source code and several simulators for non-Intel machines. The code is meant to be backward compatible with processors as old as the 8088, so if there's an old machine gathering dust in the basement, it can still be put to use for experimenting with the concepts covered in the book. The theoretical and practical parts of the book are well-balanced and easy to separate, so that one may use the book only for the theoretical parts or only for the practical parts.

Installing MINIX from the CD-ROM is not too hard, but if you have done installations before, the experience will help. The installation instructions have some minor inaccuracies, but a log file from an installation done by Andrew Tanenbaum is also provided, which clarifies the situation.

The chapters following the introduction each deal with a specific concept of OS programming. Chapter 2 is dedicated to processes, Chapter 3 to I/O, Chapter 4 to memory management and Chapter 5 to file systems. Each of these chapters starts with a description of the concept and the theory behind it. The chapter then continues to detail how the concept is realized in MINIX. This part of the chapter follows the source code line-by-line explaining what is being done by the corresponding piece of code.

Another practical side of the book is that all the code explained is contained in Appendix A, which means that there is no need for piles of printouts or a screen when one is trying to follow the description in the book.

Every chapter is followed by exercises that provide for good, practical implementation of the topics discussed.

The book ends with a suggested reading list ordered according to the concepts described in the book.



Boytcho Peytchev is a Computer Science undergraduate at the University of Wisconsin in Madison, Wisconsin. His home country is Bulgaria. He's a fan of Linux, Star Wars, Shakespeare, beer and pubs, hiking and backpacking, music, theater and chocolate. He can be reached via e-mail at bdpeytch@students.wisc.edu.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

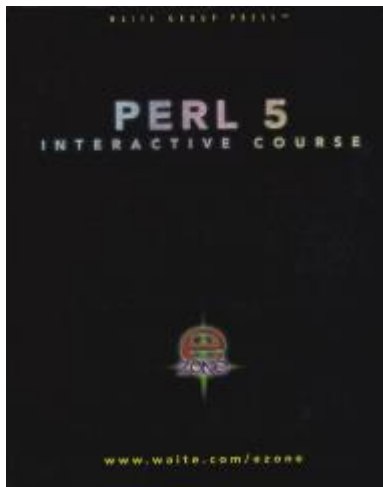
Advanced search

PERL 5 Interactive Course

Michael S. Hines

Issue #43, November 1997

Jon Orwant's vast experience with Perl is reflected in the book.



- Author: Jon Orwant
- Publisher: The Waite Group Press, Inc. 1996 (a division of SAMS Publishing)
- Price: \$49.99 US, \$70.95 Canadian
- ISBN: 1-57169-064-6
- Reviewer: Michael S. Hines

Whether you buy books by the pound (2 pounds), by the shelf inch (1 7/8 inch), for the included materials, for a unique approach to learning or for the content, the *PERL 5 Interactive Course* will meet your needs:

- Each copy includes a CD-ROM of tools and programs.
- It integrates your learning with Internet testing and assessment and provides an extensive reference to Internet resources for PERL programming.

- It provides wide coverage of the subject from getting started to advanced topics such as calling C from Perl and using Perl with C.

Jon Orwant is a Motorola Fellow at the MIT Media Laboratory and is the Editor-in-Chief of *The Perl Journal*. His vast experience with Perl is reflected in the book.

The practical extraction and report language (Perl) was designed by Larry Wall using concepts and constructs from sed, awk, C and Unix shell tools.

PERL 5 Interactive Course is one of eight books from Waite Group providing interactive web-based learning assessment (the other titles are: *HTML 3*, *C++*, *Visual Basic Scripting Edition*, *Java*, *Visual Basic 5*, *JavaScript* and *Office 97*) and on-line resources through the eZone at <http://www.waite.com/ezone/>. Buying the book gives you free access to *The PERL 5 Interactive Course* on the eZone.

The book's fourteen chapters and eleven Appendices, with reference material, teach Perl programming using Perl 5.

Each chapter contains eight training sessions with reading material and examples on the topic, a quiz and graded exercises (easy, medium, moderate and difficult) to reinforce the learning process. The chapters are:

1. Meeting Perl: The Basics
2. Pattern Matching: Regular Expressions
3. The Outside World: Files and Filehandles
4. Divide and Conquer: Subroutines
5. Mincing Words: Hashes
6. Looks are Everything: Formats
7. hOOPla: Object-Oriented Programming
8. Browsing the Libraries: Modules
9. Pushing the Envelope: Process Management
10. When Good Programs Go Bad: Debugging
11. Lock and Key: Security and Databases
12. Surfing on Oysters: Perl and the World Wide Web
13. Spreading the Word: Networking
14. Tower of Babel: Perl and Other Languages

The pace of each chapter is aggressive. For example, Session 1 of Chapter 1 begins with the classical programming assignment—"Hello World", and by Session 8 of Chapter 1 the subjects of scalars, conditional statements, loops and arrays have already been covered. (A minor editing error—the legends for

the figures in Chapter 1 are off by two—the caption for Figure 1-1 is more appropriate for Figure 1-3, for example.)

PERL 5 Interactive Course contains a wealth of reference information in the appendices. The appendices are:

- A. Quiz Answers
- B. Internet Resources
- C. The ASCII Character Set
- D. Unix Signals (incorrectly identified as Symbols in the Table of Contents)
- E. File Tests
- F. Debugger Commands
- G. Operator Precedence
- H. Command-Line Flags
- I. Perl Special Variables
- J. Regular Expressions
- K. A Summary of Functions

(A minor editing error—all references in the text to appendices is off by one—references to Appendix I actually refers to Appendix J, for example.)

The CD-ROM contains all the scripts for the projects developed in the book (over 400 programs) as well as the following tools:

- Perl 5 binaries or source for Unix, Windows 95 / Windows NT and DOS/Windows 3.x
- Contents of the Comprehensive Perl Archive Network (CPAN)—tools, scripts, resources and other material. (Note: may not be readable by DOS/Win3x systems.)
- Netmanage Chameleon Sampler (free for 30 days, thereafter the copy must be registered for \$50 US—you must make arrangements with an ISP for Internet access). The book fails to mention that you only need Netmanage Chameleon with DOS/Win3x, since all other operating

systems (OS) come with Internet Access Tools as an integral part of the OS.

- Sample of eZone content available on the Web (you'll need Netscape 2.0 or Microsoft Explorer 3.0 or later to view).

There is a section before Chapter 1 on how to install the tools and set up your system to effectively use the PERL 5 Interactive Course learning material. There is a section for each major OS (Unix, Win95/WinNT and DOS/Win3x). I tested the CD-ROM on a Windows NT 4.0 Workstation system using the FAT16 file system. The INSTALL.BAT program provided failed to complete the install properly. I was able to manually install the CD-ROM on my system performing the following steps:

1. I created a directory named "perl" in my root directory.
2. I created a directory named "bin" in the perl directory.
3. I copied the file from the CD-ROM directory /perl5/win95-nt/091-i86.zip to my root directory, and then installed it with the command **PKUNZIP -d C:\perl** (to preserve the zipped directory structure).
4. I copied the file from the CD-ROM directory /perl5/win95-nt/nt5091sr.zip to my root directory, and then installed it with the command **PKUNZIP -d C:\perl**.
5. I copied the file from the CD-ROM directory /perl5/win95-nt/ntperl5.091.ppc.zip to my root directory, and then installed it with the command **PKUNZIP -d to C:** (perl is already in the path).
6. I copied the programs perl.exe and perlglob.exe from c:\perl\5.001\src\ntt to the c:\perl\bin directory.
7. I updated my path to include the c:\perl\bin path for the Perl program and added a new environment variable, **PERL5LIB=c:\perl\5.001\lib**, to pick up the "require" library files.

Perl comes with a test script (c:\perl\5.001\src\ntt est.bat) to assure that the install was performed properly and the Perl interpreter is functioning properly. I found that this script did not execute properly as installed, but with the skills I learned from the *PERL 5 Interactive Course*, I was able to make the script work properly and check out the install. The test routine is not documented, but there is a verbose option (**-v**) which produces very detailed output during the testing phase (learned this from reading the code).

The on-line eZone resources (<http://www.waite.com/ezone/>) guide you through the registration process (**Initiate** icon), course enrollment (**Learn** icon), monitor course progress (Chapter and Session tests, and progress tracking on exams) and award final Certificate of Completion. Registration authentication is done by entering a particular word from a particular page of the book. The on-line

quiz asks the same questions included in the book at the end of each session—the advantage of on-line access is automated scoring and progress tracking. The on-line scoring must be used to obtain the Certificate of Completion. The eZone also provides access to a Mentor area for your course, where you can get personal help (**Mentor** icon) through existing Frequently Asked Questions (FAQs) or by posting your question. In addition the eZone provides a Tools and Resources section (**Chill** icon) for your course. The tools and resources available are:

- a mailing list for the course
- a reader newsletter
- resources on the subject matter

I did access and print the readers' newsletters, *The eZone Newsletter*, for February (14 pages), March (16 pages) and April (15 pages) 1997. The newsletters keep you informed of developments on the eZone, new courses and provide a summary of key discussions from the mailing lists. The April 1997 ezone Newsletter reports that these courses are now available for Continuing Education Units (CEUs) through Marquette University (<http://www.mu.edu/>).

The web site also includes the Perl 5.0 eZone Club House which archives past mail on the mailing lists and makes it available for searching and access. A few of the vast Perl resources linked from this page that I checked out were: The Perl Language, Index of CPAN (the Comprehensive Perl Archive Network), Perl 5 WWW page—Perl 5 Information, Announcements and Discussion and the HTML version of Rex Swain's *Perl 5 Reference Guide* (23 pages).

Overall, I would rate this book as a best buy—the comprehensive coverage of the uses of Perl in this one volume would make this a frequently used desk reference. I have seen similar material, but it has been in several different books that address particular applications of Perl. This book provides a wide assortment of applications of Perl in one place.

Michael S. Hines is the Senior Information Systems Auditor for Purdue University, West Lafayette, Indiana and has a personal interest in Computer Security. He holds the Certificate in Data Processing (CDP) and the Certified Fraud Examiner (CFE) professional designations. He has a Bachelor of Science in Engineering and a Master of Science in Computer Science degrees, both from Purdue University. He tinkers around with Linux, FreeBSD, BSDI, SunOS, Sun Solaris, AIX, HP-UX, MS-DOS, Win3.x and Windows NT operating systems. He is a member of the Purdue Computer Emergency Response Team (PCERT). He has presented training to various audiences on “Audit Use of the Internet”, “Internet Security” and “Auditing Routers and Firewalls”. Michael can be contacted by e-mail at mshines@purdue.edu.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

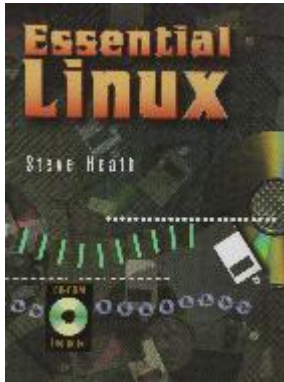
Advanced search

Essential Linux

Marjorie Richardson

Issue #43, November 1997

The book is well written and clear, although like most books it could have benefitted from one more proofreading pass.



- Author: Steve Heath
- Publisher: Butterworth-Heineman
- Price: \$39.95 US
- ISBN: 1-55558-177-3
- Reviewer: Marjorie Richardson

Essential Linux is written to give you an understanding of the basics of Linux using the Slackware v3.1 distribution. It is directed to the newcomer, the same market as Matt Welsh's *Linux Installation and Getting Started* and O'Reilly's *Linux in a Nutshell*. I think the fact that so many books directed toward the Linux user seem to be appearing on bookshelves these days is a sure sign that publishers are recognizing the growing market to be found for Linux.

The book describes author Steve Heath in this way:

“The author has been involved in the design and development of microprocessor-based systems since 1982. He is the author of 15 books in the computing field.”

As described, Mr. Heath is certainly well qualified to write a Linux book for the novice. The book is well written and clear, although like most books it could have benefitted from one more proofreading pass. I found it a bit annoying that the word megabytes is abbreviated as Mbytes instead of MB, but certainly not confusing. In the trade name declarations, Mr. Heath prematurely announces Linux as a trademark of Linus Torvalds. (Certainly a statement we hope is true by the time this review is printed.)

Chapters are arranged nicely, starting with the introductory material (installation, booting, etc.) and going on to discuss commands, shell scripts and editors. Finally, there are short chapters on system administration, networking and installing XFree86.

The installation chapter is comprehensive and accurate, working its way from setting up the keymap and disc partitions, through software installation and configuration, to building and testing the kernel. There are also discussions of LILO, kernel patches and the /proc file system. All of this is good information that is necessary in order to get up and running with Linux.

The basic Linux commands are also covered well. Mr. Heath uses a standard format to present the information, which includes the command definition, the arguments and an example. While he is quite explicit about the dangers of using the remove command (**rm**) carelessly, he neglects to offer the usual advice to the novice of setting up an alias for **rm** containing the **-i** option, so that the user is always queried for confirmation before removing a file.

The bash and tcsh shell chapters are adequate and full of examples to aid the novice in writing scripts of his own. No mention is made of the korn shell, which some of us still do use.

Editors are given short shrift. Mr. Heath concentrates on the ed and ex line editors, giving 5 of some 20 pages to vi and ignoring Emacs entirely.

The chapters on system administration and networking are also short given their subject matter, but they do give a good introduction to the basics for both—and this is all the title has promised.

The final chapter is entitled “If It Doesn't Work” and gives some solutions to a couple of common problems and some tips for debugging shells. At less than five pages, it is the shortest chapter in the book. I guess not much goes wrong with Linux.

Essential Linux comes with a CD-ROM that contains Slackware v3.1, documentation from Sunsite and various versions of the Linux kernel from 1.3.2 through 2.0.06 and 2.1 (development).

In conclusion, if you are a Linux beginner who has chosen Slackware as your distribution, *Essential Linux* is one of many books that can help you get started. It amply fulfills its promise of presenting the *essentials*.



Marjorie Richardson is the Editor of *Linux Journal* and the on-line e-zine *Linux Gazette*. In a past life she was a computer programmer for the oil industry who wrote geophysical applications. She enjoys outside activities, motorcycling, quilting, movies and science fiction. She can be reached via e-mail at info@linuxjournal.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Linux Universe, 3rd Edition

Jan Rooijackers

Issue #43, November 1997

A quick scroll through the book gives the impression that a lot of time was spent to put it together.

- Authors: S. Strobel, R. Maurer and S. Middendorf
- Publisher: Springer-Verlag
- E-mail: custserv@springer-ny.com
- URL: <http://www.springer-ny.com/>
- Price: \$34.95US
- ISBN: 0-387-94879-1
- Reviewer: Jan Rooijackers

Linux Universe contains 2 CDs: CD 1 contains the source code (kernel versions up to 2.0), and CD 2 contains the installation files. A quick scroll through the book gives the impression that a lot of time was spent to put it together. It has a nice overall layout (including the cover), and catchwords are printed in the left or right margin. My first impression of the book was good; I was ready to discover what the text would bring me.

The book contains nine chapters and a reference section. It begins with a brief history of Linux that is not so deep as to become boring for a new Linux user. Chapter 2 consists of 2 pages and gives a summary of the most important features of the Linux distribution included on the CD-ROM: X Window System Version 6, GNU utilities and programs, a complete Unix development environment and many other tools. One important thing to mention is the smart CD-ROM cache—more on this later. A 1-page chapter follows to explain the new features in Linux 2.0. Chapter 4 explains to the reader which hardware is required to run Linux and includes a long list of supported hardware—from SCSI adapters and sound cards to PCMCIA adapters. The ATI Mach 64 video adapter is supported, but I still had problems with it and with the Slackware release.

Prior to the installation chapter, there is a chapter which explains the various methods of installation: a fixed installation or an installation using the smart file cache. The smart file cache method keeps the software on the CD until it is invoked for the first time, then the file cache copies it to your hard disk. On the next invocation, the program starts directly from hard disk. As a result, your hard disk will only be filled with software that you actually use. The same chapter explains how to partition your hard disk.

In chapter 6 the installation procedure is presented. The reader is guided through the installation via very clear pictures of the graphical user interface (GUI). Additional boot options are explained in Chapter 7 of the book. During my installation of the distribution, I had problems with the configuration of my ATI Mach 64 video card; to solve this I made use of the Metro-X software already in my possession. I think it would be nice for the next release to include Metro-X, particularly since it is usually included in the Slackware distribution anyway.

The next 2 chapters describe the fundamentals of Linux and system administration. In the fundamentals chapter, the reader learns how to mount a CD, change directory, change permissions for file/directory, copy, move and remove, print and many other techniques. If the reader has a problem, he can invoke the on-line documentation, which is included on the CD as well. It is not necessary to read the whole chapter to get to a specific topic, you can just jump to the desired paragraph.

The administration chapter defines administration and explains the powerful **xadmin** program. The book makes use of this program for all administration jobs. Also in this chapter, the configuration of X11 is explained—from a detailed description of the XF86Config file to the configuration of the window manager (FVWM) file.

The reference section is the last part of the book. This section explains a lot of Unix commands with their associated options clearly and concisely.

Linux Universe offers a useful start for the newcomer to the Linux world. The text is clear and very readable; the examples are good and well explained with figures. The GUI (graphical user interface) makes it very easy to install this distribution, and the xadmin program brings a lot of benefits. All of the installation documentation and the publications of the Linux Documentation Project are present. For me, the best of all is the ability to install Linux via the cache mechanism.



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

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

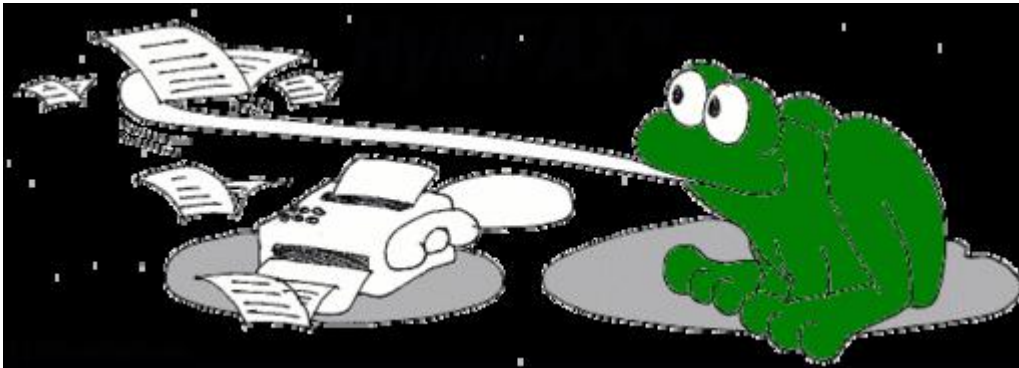
[Advanced search](#)

Faxing From a Web Page

David Weis

Issue #43, November 1997

HylaFAX and Linux are used to provide a powerful, freeware alternative for faxing from a Mac network.



Part of my job at the publishing company where I work is finding new ways to finish time-consuming jobs more quickly. After implementing an on-line order entry system and a Mac file server, I set up an e-mail/dial-up router for a remote office. The staff did a lot of faxing which consumed a huge amount of time that could have been better spent doing other tasks. The small 486 we used didn't have enough horsepower to handle fax conversion, so I didn't try to use it for that job. After the power supply fan quit and the machine died, I decided to get a more powerful machine and set up a fax server.

We ordered a 133 MHz Pentium 5 for the new server. It has an Ethernet card and two modems to eliminate fighting for the telephone line. One modem is used for the PPP dial-up, and one is a Class 2.0 fax-modem. Most fax-modems implement only Class 1 fax transmissions, which require the computer to do most of the processing, but Class 2.0 frees up those resources.

I had used the HylaFAX software a few years before and knew it could easily handle the volume of faxes that we send. More importantly, it was within my budget. By using Linux and other free software, I had the money for hardware

and another phone line—money that would have otherwise been spent on expensive software.

HylaFAX uses Ghostscript to convert faxes from PostScript to the tagged image file format (tiff), as faxes are sent over the line in a form of tiff. One of the most important things to do is make sure that the Ghostscript software is set up correctly and has the tiffg3 driver enabled. Without this driver, you will spend a long time wondering why you can't finish the HylaFAX setup.

Originally, I just wanted to be able to fax a certain set of documents. Part of the way through the setup process, I found a Macintosh extension called MacFlex that allows a Mac to print to the fax server as if it were a printer. After you choose print, MacFlex asks for the phone number and spools the document to the server. It is well-written software that has produced more “oohs” and “ahhs” in the office than the server, and it is available from <http://www.eats.com/>.

In order for the users to send documents to the fax server from their own Macintoshes, Netatalk was installed. Netatalk lets you share Linux drivers with Macintosh machines. These drivers show up in the Chooser like any other server.

After installing and testing Ghostscript and finding the location of the executables and fonts, it's time to get HylaFAX. Download information and documentation is at <http://www.vix.com/hylafax/>. Most of the configuration consists of setting paths to reflect the directory structure of your system. After the software is installed, another round of configuration is done to set up the modems. First, the program **faxaddmodem** probes the modems to determine the manufacturer, speed, class and other information. Then, in a send-only environment, the program **faxmodem** is run to tell the daemon that a new modem exists. I could never get the **faxmodem** program to work correctly, so I ran the program **faxgetty** on the correct serial port. **faxgetty** is a specialized version of **getty** that reports status information on the modem to the fax daemon.

The program included with HylaFAX to queue outgoing faxes uses a command-line interface. Since the users of the system were accustomed to Macintoshes, I didn't think they should have to learn Unix just for the purpose of sending their faxes. Therefore, I wrote a Web front-end to HylaFAX that allows them to select one or more documents from a list and choose to send it to a distribution list or to an individual.

After getting HylaFAX completely installed and running, I thought the worst was over. However, HylaFAX comes with a default cover page that includes the Silicon Graphics logo, and we already had a nice cover page I wished to

continue using. I know a small amount of PostScript, but 1 megabyte of machine-generated PostScript is nearly impossible for me to decode. After trying unsuccessfully for half a day to change the default to match our cover page, I looked through the information on the HylaFAX web site mentioned above and found more helpful instructions on constructing the cover page. I removed a graphic from the default cover page that cut the size of the PostScript file from 1MB to about 100K. This version was much easier to modify, and I had it going in about an hour. The cover page on the web site is a template that the program **faxcover** fills in. In order to put in the place holders that faxcover needs, you must open the PostScript version of the cover page in a text editor and insert the correct fields. Checking the cover page for appearance is difficult, since the blanks show up empty in a PostScript viewer unless you first run faxcover and put in some test data. Doing this just makes a version of your cover page for you to look at—it does not send any faxes.

Overall, the project was a success. After a few minor problems, the system is chugging away handling IP masquerading, UUCP over TCP and out-bound faxing. The total hardware cost was under \$2000US. I spent nearly 60 hours writing the front-end software(1) and making sure all the other pieces worked together correctly. The time required to send a fax dropped from 10 to 15 minutes to about 30 seconds.



David Weis weisd3458@uni.edu, is a computer science student at the University of Northern Iowa. His favorite things to do include spending time with his girlfriend and solving problems using Linux.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

The UNIX Web Server Book

Gerald Graef

Issue #43, November 1997

This book is well-written and informative, fills a gap in available books, and its authors bear impeccable credentials.



- Author: R. Douglas Matthews, Paul Jones, Jonathan Magid, Donald A. Ball, Jr., Michael J. Hammel
- Publisher: Ventana Communications Group, Inc.
- <http://www.vmedia.com>
- ISBN: 1-56604-480-4
- Price: \$49.99US
- Reviewer: Gerald Graef

The most overly published book today is the WWW book. How to, Where to, Why to. It's all available in repetitious detail. Only a few aspects of the Web aren't overdone. Server administration is a big one (a WWW book catalog is another). A good book with background and information on the various choices of server software would be a boon. Such a book is *The Unix Web Server Book* by R. Douglas Matthews, Paul Jones, Jonathan Magid, Donald A. Ball and Michael J. Hammel. It is well-written and informative, fills a gap in available books, and its authors bear impeccable credentials.

Setting up a web server is not a difficult task (given a stable Internet site), but it is one that requires a unique set of tools. *The Unix Web Server Book* was

written to allow someone with little or no experience with the Web to set up a server. This is perhaps something of an anomaly, since anyone capable of administering a Unix system is likely to have enough basic knowledge to configure a web server using software documentation. Still, there is a necessary background that this book attempts to fill.

The Unix Web Server Book assumes no prior knowledge of the World Wide Web and so begins with a brief history. Chapter two, "The Basic Pieces", is summed up by the chapter legend "What is this http:// stuff?". Chapter three compares servers and recommends Apache, a freeware server available for many platforms. An introduction to web security follows. In the next ten chapters, introductions are given to the elements of developing web pages, including HTML, importing documents, images, multimedia, searching, forms, advanced CGI, Java and HTML checkers. Finally, "Fitting in: Joining the Virtual Community" finishes up the book. It's all at a pretty basic level, but the sum is a hefty book with voluminous references to on-line resources.

Included with the book is a CD-ROM containing many of the tools and programs necessary to get a web server up and running, including the Red Hat Linux distribution, Apache web server, HTML utilities, graphics programs, CGI packages and more, all arranged by chapter of discussion. The collection makes an excellent beginning to a web site ("beginning" since no true webhead is ever content).

When I received a copy of the book, I was hopeful of an easy installation of the latest version of Red Hat Linux. Unfortunately, the version included is not supported by either Red Hat Inc. or the publisher of this book. Moreover, the CD-ROM directory structure is incorrect; although the distribution is complete, the install program expects a specific directory structure that is not adhered to on the CD-ROM. It is possible to copy the files onto a hard disk (about 300MB total) and install. However, a better option is to splurge and buy the latest version of a Linux distribution. This has the added benefit that many of the CD-ROMs available contain software useful for web sites.

The Unix Web Server Book is a solid guide for someone new to the Web who is charged with developing not only a web site, but also a web server. More advanced readers may find the book too basic and may often be frustrated at the tantalizing hints of deeper truths. Fortunately, the book is a gold mine of annotations to web resources for everything covered—right back to Vannevar Bush's ground-breaking 1945 article "As We May Think" on the electronic linking of documents. Future server administrators may find this book useful depending on their network experience. More likely, they'll gain more from a book that is dedicated to whatever task is at hand. Finally, one group who will find this book of interest is the experienced web surfers looking to understand

how the Web works, but who aren't actually going to set up a server. And don't overlook that CD-ROM—if you don't have fast access to the Internet, having all the basic web software at your fingertips is very nice indeed.

Gerald Graef is a doctoral candidate in theoretical physics who lives on Alpha, Sparc, and Linux computers. He welcomes comments and questions at ggraef@usd.uwm.edu or through his home page at <http://www.uwm.edu/~ggraef>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Integrating SQL with CGI, Part 2

Reuven M. Lerner

Issue #43, November 1997

This month we learn additional ways to retrieve data from a relational database and ways to divide our data into multiple tables for maximum efficiency.

In the last installment, we looked at ways to use a relational database server from within CGI programs. Relational databases are a great help, since they provide us with a relatively simple means of storing and retrieving information.

Everything in a relational database is stored in two-dimensional tables, with each row representing a record and each column a field within that record. When we use ASCII text files to store our information, we generally have to write our own routines for saving and retrieving that information; by structuring our information in tables, we can save ourselves quite a bit of time and energy, as well as ensure that our data will not be corrupted when multiple instances of our program attempt to modify the information.

Communication with a relational database is carried out using the Structured Query Language (SQL). Because SQL is not a programming language, we have to embed our SQL queries inside of a C or Perl program if we want to execute them.

Last month, we looked at a set of CGI programs which allowed visitors to our web site to send and receive web-based postcards. Users entered information about themselves and their postcards' recipients, along with the name of a graphic and a short text message. Each postcard was stored in the postcards table, with each row in the table representing a single postcard. By giving each row a unique ID number, our program was able to retrieve individual postcards from the database.

Indeed, the program `send-postcard.pl` that we examined last month demonstrated how it is possible to take inputs from an HTML form and turn them into a postcard stored in our database, as well as how to take that unique

ID number and use it to display a postcard with `show-postcard.pl`. The unique ID number is generated by `send-postcard.pl` when it creates a new row in the `postcards` table and ensures that users can retrieve their own postcards while making it relatively difficult to find those addressed to other people.

The programs we are examining are written in a portable enough version of SQL that they should run on just about any relational database server. However, I have only tested these programs using MySQL, a nice, small relational database product for several versions of Unix, including Linux. You can find out more about MySQL on the Web at <http://www.tcx.se/>.

Replacing Names With Numbers

One of the problems with that version of `send-postcard.pl` was that it asked the user to name a graphic file to be inserted into the postcard she was sending. While this sounds like a nice idea, there were several problems with it, most notably the fact that the user could name a file that did not exist on the server. Moreover, there was no way for the user to know which graphics did exist on the server, short of writing another program that would present a directory listing.

Additionally, storing the file names for the graphics alongside the other postcard information is inefficient and can lead to synchronization problems. After all, what happens if you decide to change the name of a file from `foo.gif` to `bar.gif`? Under our current model, we would have to go through the entire “`postcards`” table and rename each row in which `foo.gif` appeared. This is not terribly difficult to do, but it means we have to think carefully before making such a change.

Finally, storing the names of the graphics files introduces the possibility of typographical errors. A user might inadvertently type `foo.gif`, and thus prevent the postcard's recipient from seeing the desired graphic.

In order to solve all of these problems, we will move the names of the graphics files to a separate table, giving each graphic a unique ID number, much as we did for each postcard. With this system in place, we can then refer to graphics by number in the `postcards` table. We can still refer to an infinite number of graphics in this way, but we do so by using an integer, rather than a string, saving storage space on our file system. In addition, such consolidation allows us to change one or more of the graphics file name(s) or location(s) without having to worry about ruining our program's output.

Creating our Graphics Table

We can retrieve one or more rows from a table in our database using the SQL **SELECT** command, which returns all of the rows matching our selection criteria. Thus, if we are interested in retrieving the row with the ID number of 12345 from the postcards table, we can do so by sending the query:

```
select * from postcards where id_number = 12345
```

This will return a small table, the rows of which all have an `id_number` equal to "12345". Since `id_number` is guaranteed to be unique, we can expect that the table returned by the **SELECT** command will consist of a single row and all of the columns contained in the postcards table. If `id_number` was not guaranteed to be unique, then **SELECT** might well return a number of rows from the table, rather than only one.

We can wrap SQL queries inside of programs, and CGI programs are full-fledged programs; thus, we can easily write a CGI program that:

1. takes an argument from a user,
2. uses that argument in the **SELECT** statement sent to the database server, and
3. gives us a new **SELECT** statement each time we run the program.

Let's create a new table, in which the first column uniquely identifies the graphic, and the second column names the graphic. I do this using the interactive **mysql** program that comes with MySQL, which I enter by typing **mysql<\!s>test**. To create the new table, enter the following lines:

```
create table graphics (  
    graphic_id mediumint not null primary key,  
    graphic_file varchar(60) not null)
```

The interactive **mysql** program indicates that the command was executed successfully by giving us the following message:

```
Query OK, 0 rows affected (1.13 sec)
```

We have now created a table with two columns. The first, `graphic_id`, stores medium-sized integers, cannot be null and must be unique ("primary key" in SQL lingo). The second column stores a variable-length string containing up to 60 characters and must be non-null.

We can now insert a graphic into the graphics table with the following command:

```
INSERT INTO graphics (graphic_id,graphic_file)
VALUES (12345, foo.gif);
```

Now if we ask for the contents of the graphics table, we get the following:

```
mysql> select * from graphics;
1 rows in set (0.04 sec)
+-----+-----+
| graphic_id | graphic_file |
+-----+-----+
|      12345 | foo.gif      |
+-----+-----+
```

We have now proven it is possible to store a record of a particular graphics files, as well as to retrieve the name of that file based on a unique ID number. Now comes the hard part—making an association between the ID and the graphics file name.

First of all, we need to modify the definition of our postcards table, such that it now expects to get the ID number of a graphic, rather than the graphics file name. To create the postcards table, use the following SQL command:

```
create table postcards (
    id_number mediumint not null primary key,
    sender_name varchar(60) not null,
    sender_email varchar(50) not null,
    recipient_name varchar(60) not null,
    recipient_email varchar(50) not null,
    graphic_id mediumint null,
    postcard_text blob);
```

Notice that we have removed the `graphic_name` column from the table we defined last month and replaced it with a column named `graphic_id`—which has the same name, size and type in both of the tables. We don't need to give the columns identical names, but doing so makes it easier for us to keep track of things.

The most straightforward way to replace the old `graphic_name` column with the new `graphic_id` column is to create the table anew, as demonstrated with the above SQL command. However, this action results in the loss of any data already in our table. In the beginning of a project that uses a database, creating and destroying tables is quite normal—at least until you get the definition right. After a table has been used for a while, you probably don't wish to destroy your data.

To solve this problem, we simply add our new column (`graphic_id`) to the `postcards` table using the **alter table** command, which allows us to add one or more columns to an existing table. This is not the most efficient way to design a database, but it eliminates worry about programs which might be dependent on `graphic_name`.

Joining two tables

Our database now contains two tables: "postcards", with all of the information required to assemble a complete postcard (including the ID number of the graphic we wish to display), and "graphics", which maps those ID numbers to the names.

How do we create an association between the two? We already know that it is possible to use SQL's SELECT command to retrieve one or more columns from a table. SELECT can also be used to retrieve columns from multiple tables and can even join columns together from them.

For instance, let's get the sender name, recipient address and graphic name from our two tables, such that the graphic name for each postcard is displayed immediately next to the unique ID and recipient address.

First, let's add another graphic file to the graphics table, just for good measure:

```
mysql> INSERT INTO graphics (
      graphic_id,graphic_file)
      VALUES (67890, bar.gif);
Query OK, 1 rows affected (0.00 sec)
mysql> select * from graphics;
2 rows in set (0.16 sec)
+-----+-----+
| graphic_id | graphic_file |
+-----+-----+
|      12345 | foo.gif      |
|      67890 | bar.gif      |
+-----+-----+
```

Let's now create a postcard to use for our example:

```
insert into postcards (id_number, sender_name,
      sender_email, recipient_name,
      recipient_email, postcard_text, graphic_id)
VALUES (99999, "Bill Clinton",
      "president@whitehouse.gov",
      "Al Gore",
      "vice.president@whitehouse.gov",
      "Please call. I have a new tax idea.",
      12345)
```

Notice that because they are integers, neither the postcard ID number nor the graphic ID number is surrounded by quotation marks. Also, notice how the initial list of columns in our INSERT command includes graphic_id, but leaves out graphic_text. Since graphic_text in postcards can contain null values, we can ignore it when inserting new rows into the table.

Now let's retrieve the sender name, recipient name and graphic file name for the postcard we just created:

```
mysql> select postcards.sender_name,
      postcards.recipient_name,graphics.graphic_file
-> from postcards,graphics
```

```
-> where postcards.id_number = 99999;
2 rows in set (0.31 sec)
+-----+-----+-----+
| sender_name | recipient_name | graphic_file |
+-----+-----+-----+
| Bill Clinton | Al Gore       | foo.gif      |
| Bill Clinton | Al Gore       | bar.gif      |
+-----+-----+-----+
```

Egad—this isn't what we wanted at all. We wanted it to bring up the file name for the ID we specified, not the combination of all rows and columns in the “graphics” table with our postcard. That's what we wanted to do, but that's not what we told the computer to do. By formulating our SQL query as above, we inadvertently asked for all possible combinations of rows from postcards with rows from graphics, this combination is known in database circles as the “Cartesian product” of the two. While asking for a Cartesian product is not an error, it is almost always undesirable. When dealing with especially large tables, asking for such a combination of tables can result in a long, unnecessary computation that ties up the database server keeping it from performing other tasks.

How can we modify our database query so that it does what we originally wanted, namely, giving us the graphics file name in place of its ID number in the postcards table? The simplest way is to set up a restriction between the two tables, adding to the WHERE clause in the query, as follows:

```
mysql> select postcards.sender_name,
postcards.recipient_name, graphics.graphic_file
-> from postcards, graphics
-> where postcards.id_number = 99999
-> and postcards.graphic_id =
graphics.graphic_id;
1 rows in set (0.10 sec)
+-----+-----+-----+
| sender_name | recipient_name | graphic_file |
+-----+-----+-----+
| Bill Clinton | Al Gore       | foo.gif      |
+-----+-----+-----+
```

Now that's more like it. By requiring the equivalence between the graphic_id columns in both tables, we retrieved the information as if it came from a single table. That's part of the magic of SQL and relational databases. By combining tables in this way, you can make your data easier to handle by putting it in separate tables. However, when you retrieve the information, no one knows it came from separate tables, since a new, temporary table is returned to the caller.

This may seem like a silly example, but imagine a corporation with very strict pay scales whose payroll is handled by a relational database. If you give every employee an ID number indicating salary, you can give everyone a raise (or a reduction, depending on the company's financial state) by updating records in a salary table, rather than the table of employees. The next time you perform a

join between the employee and salary tables, the new salary will be reflected automatically.

Turning the SQL into Perl

Now that we have seen how to get our queries to work at the SQL level, let's think about the necessary steps needed to integrate these queries into some CGI programs. For the most part, our CGI programs do not need many changes. We need to modify `send-postcard.pl` so that it inserts the graphic ID into the `postcards` table, rather than the graphics file name, and `show-postcard.pl` needs to use the SQL query that we formulated above in order to get the graphic file name from the `graphics` table in addition to the information in the `postcards` table. The revised versions of the code are not completely reprinted this month. These two listings along with the listing that is printed are available by anonymous download in the file <ftp://linuxjournal.com/pub/lj/listings/issue43/2508.tgz>.

First, we'll look at the revised version of `show-postcard.pl`. The only change to be made to the listing printed last month is in the SQL query, which now reflects the new table:

```
my $command = "";
$command = "select postcards.sender_name,";
$command .= "postcards.sender_email,";
$command .= "postcards.recipient_name,";
$command .= "graphics.graphic_file,";
$command .= "postcards.postcard_text from ";
$command .= "postcards,graphics ";
$command .= "where id_number = $id";
$command .= "and postcards.graphic_id = ";
$command .= "graphics.graphic_id";
```

Only this one change is necessary, because of the way in which we wrote the original version of `show-postcard.pl`. By contrast, imagine how much code we would have needed to rewrite if we had initially stored the information in a single ASCII text file, and then split the information between two files.

Our modifications to `send-postcard.pl` is almost as easy. We need to add the definition of `$graphic_id`, rather than `$graphic_name`, at the top of the file:

```
my $graphic_id = $query->param("graphic_id");
```

When we insert the postcard into the `postcards` table, we must modify the code so it uses the `graphic_id` column and variable, rather than `graphic_name`:

```
$command = "insert into postcards ";
$command .= " (id_number, sender_name, ";
$command .= " (sender_email, recipient_name, ";
$command .= " recipient_email, graphic_id, ";
$command .= " postcard_text) ";
$command .= "values ";
$command .= " ($id_number, \"$sender_name\", ";
$command .= " \"$sender_email\", ";
```

```
$command .= " \"$recipient_name\", ";  
$command .= " \"$recipient_email\", ";  
$command .= " \"$graphic_id\", ";  
$command .= " \"$postcard_text\" ) ";
```

With those modifications in place, we are done. Now our code will work just fine with the new table, storing and retrieving graphics according to their ID.

Creating the form

There is one remaining problem with this version of the code. How is a visitor to our site supposed to know or remember the ID numbers for the various graphics that are available? We could modify the HTML form to provide this information, but it seems a bit silly for us to do so, since we would then have to update the form each time we updated the table.

The simplest solution is to write a small CGI program that produces the HTML form, inserting the values as appropriate. There are a number of different ways to allow the user to choose, but I decided when writing this program to take a relatively easy path by using radio buttons. A more aesthetically minded programmer (or one who expected to have a lot of graphics files) may have chosen a selection list, but that's a side issue. The resulting program, `postcard-form.pl`, is shown in [Listing 1](#).

That about does it for our postcard-sending problem. There are, of course, many other ways in which this set of programs could be extended or modified. For example, it might be a good idea to create a CGI program that would allow us to enter and edit the file names in the graphics table, so that we would not have to use the interactive mysql program for such modifications. Currently, only someone knowledgeable in SQL can add, modify and delete elements in the graphics table. We could also ensure that the ID numbers in the graphics table are given sequentially; some relational database vendors provide that facility, allowing for “identity” columns that automatically increment as new rows are added.

It would also be nice to allow users to preview the graphics they place on the postcards, or at least describe the pictures rather than just presenting the users with file names. This option might require storing two versions of each graphic or adding another column to the graphics table that would be used for descriptions or previews.

The possibilities, as you can tell, are unlimited—and this is a relatively small project.

This article ends our whirlwind tour of SQL, although future columns will undoubtedly continue to use relational databases as a means for storing

information. Next month, though, we will look at the efficiency of our CGI programs, including the “CGI lite” module for Perl.

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](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Letters to the Editor

Various

Issue #43, November 1997

Readers sound off.

Slackware to Debian

...I read the article "Is Linux Reliable Enough?", by Phil Hughes in the July 1997 issue. Mr. Hughes indicated that SSC is using Slackware 96, but is changing to the Debian distribution. Here are a few questions:

1. Why are you considering a change to Debian?; 2. Is there a problem with the Slackware 96 distribution?3. What are the advantages/disadvantages of Debian?4. What are the advantages/disadvantages of Slackware 96 and other versions?

I would also like to see articles on the following if it is possible:

1. Comparing the RDBMS products such as Solid, Empress, mSQL, Postgress and etc products.2. Comparing the Motif and other GUI development tools such as Xforms.3. Many articles stressing the important uses of Linux in the Real World —Edmund P. Morgan Senior Software Analyst Emorgan@cup.net

While Slackware has been reliable, its upgrade management tools are lacking. Debian (and Red Hat) offer an installation system that includes dependencies. In addition, Debian offers a menu choice to upgrade all installed packages. When you are managing the software on 25 systems, good configuration management can save you substantial amounts of time. We are planning to do an article comparing the various distributions next year in our June issue.

You must be reading our Editorial calendar for next year: our focus for the February 1997 issue is databases, for March, it is GUIs and for April, it is Business Solutions. In past issues we have done reviews of most of the databases as well as addressing GUI development tools; we have also covered

Tcl/Tk and Xforms and present a monthly "Linux Means Business" column. —
Phil Hughes, Publisher info@linuxjournal.com

LJ Cover on Education

I was on your web page and saw the cover for the October Education issue. I was a college professor when I felt my life most had meaning, and because of that, I always enjoy working with college students. Your cover brought that feeling home to me again.

It looks like a good issue, and I expect to enjoy reading it. —Jon "maddog" Hall
maddog@zk3.dec.com

My First Issue

Yesterday I received my first issue of *Linux Journal* in my mailbox. I have not put it down since. This is the *best* source of information for Linux I have found. I run a production Linux network/Internet site, and I am always looking for information about new tools and other Linux information. *LJ* has already helped me solve the problem of how to monitor my Linux servers. I've been using Linux since 0.99.something and have completely converted from other partial Unices. Keep up the good work. The only thing I can see that would help *LJ* is if it came out once a week instead of once a month. I can dream... —Darren Young youngd@cirrusnet.com

Bridging the Generation Gap

My parents just left with two copies of *Linux Journal* (#38 and #39, #40 appeared in the mail today)--in spite of the fact that up until now, they have only used MS-DOS for their computing needs.

In Dutch we have this proverb: "Je bent nooit te oud om te leren"--you're never too old to learn. So, who knows ... —Toon Moene, The Netherlands
toon@moene.indiv.nluug.nl

Choices

After reading Mr. Temple's letter to the Editor in the August 1997 issue, I must disagree. The popularity of Linux has exploded in the past year, not because of people who want to see how a true preemptively-multi-tasked operating system works, but because people want to use this operating system to accomplish tasks (i.e., real work).

I can find my way around a kernel; in fact, hacking the Minix kernel was required in my Operating Systems class in college. However, my main interest

in using Linux is to do my job. To that end, I've used Linux on a surplus 386 SX/20 to build a company firewall/gateway and e-mail server/router. Using IP masquerading, sendmail and a few custom Perl scripts, this throw-away machine is now responsible for routing both incoming and outgoing e-mail and for firewalling our Intranet over a single dynamic PPP line. To me and to a lot of other readers of *LJ*, networking is the main draw to Linux. One of the beauties of Linux is that it has something for everybody.

Keep "Kernel Korner"—I read it—but keep on including the networking articles. Linux is too diverse for *LJ* to resemble *PC Magazine*. —T.J. Harrell III
harrell@navajo-refining.com

Webmaster Handbook Idea

As an avid reader/subscriber to *LJ*, I've been collecting your articles related to webmastering and running a network/web server. This spans the gamut from creating/using CGI scripts to how to use "TCP Wrappers" (August 1997). I've got quite a selection now, and it's occurred to me that others may have the same need.

So, may I suggest *LJ* package reprints of all the pertinent articles from years past, add an index and perhaps some advertising, then sell it for a reasonable/nominal price, perhaps \$5.95US (myself knowing nothing about the publishing, especially the money).

As an alternate, perhaps you could put up a page on your web site with an index to the articles, with each a link to a summary page or the actual text, if available. This might encourage people to then buy the back issues. —Scott Daniel sscott@blueghost.npt.nuwc.navy.mil

I'll put it on the "wish" list. We do put articles on the web site after a time, but it's never certain as to when someone will have the time to do the HTML. We are also planning to start putting past issues on CD-ROMs and sell them in that format. The 1996 issues are in the works now.

Lotus Notes for Linux

Half a year ago, I'd asked on the Caldera mailing list what kind of applications people would want on Linux that weren't yet available. The two most-requested applications were an accounting package and Lotus Notes. With the Appgen announcement, one of those is now covered. As for the other...

I was in conversation recently with someone who has been talking to IBM (at an appropriately high level) about porting Notes to Linux. That person told me that *any* testimonials about potential end-users who would buy Notes for Linux

would be instrumental in convincing IBM to make it happen. There are no technical issues; this is only a matter of marketing and convincing the sales folk.

While I have no use for Notes, I believe its availability for Linux would be both an important application and yet another validation of Linux's suitability for corporate use.

Please e-mail me any relevant information and I promise to pass it on ASAP—added to a few stories of my own... —Evan Leibovitch, Ontario evan@telly.org

Really Push League

I read Doc Searls article, "Shoveling Push Media", in *Linux Journal*, June 1997. I threw out my TV six years ago; if push ever comes to shove, I'll do the same with my computer systems, too.

I've been in this business for fun and profit since the days of CP/M, designing circuit boards, programming, etc. and on the Internet for about a decade. — Max Southall max@prninfo.com

Zombie Processes

In the August 1997 issue (just received), I have spotted what I believe to be an error. On page 40, In the column "Best of Technical Support" there is a footnote to the paragraph headed "Mysterious Zombie Process" which states, "When a parent process dies or is ended, any child process started by the parent becomes a zombie process."

The condition described is that of an orphan process, not a zombie. Orphan processes are created when their parent process terminates, and all orphans are automatically adopted by the scheduler process (process 1).

A zombie process, on the other hand, is *any* process that has terminated but not yet been cleaned out of the process table. In some of the original Unix literature from the folks at Bell Labs, they comment that a zombie process can only be resurrected by the use of arcane magic.

In the voodoo religion, a zombie is a dead person who has been reanimated via magic, so the appellation when applied to a Unix (or Linux) process is not quite appropriate, but it comes close.

As to the problem experienced by Scott (the author of the note), I would venture to guess that the process he is invoking (**netcfg**) performs a `fork()`, `exec()` combination, and the parent process then exits. This is a typical hack

used to prevent the invocation of a program from hanging up the terminal if the user forgets to use the & character to run it in the background.

Since the parent has terminated, the terminal is again available for command input. The child process will still be running in the background, however. For whatever reason, the parent process is not able to be cleaned out of the process table (perhaps until the child processes all terminate?), and so it shows up as a zombie. —Harry Gross harry.gross@ix.netcom.com

The footnote in question is almost a direct quote from “Unix: An Open Systems Dictionary”, William H. Holt and Rockie J. Morgan, Resolution Business Press, 1994. This dictionary also defines an orphan process as being synonymous to a zombie process. I wanted something short for the footnote, and the Dictionary gave me that sentence—perhaps, I need to buy an updated version.

U.S. Domain

In regard to the article, "Registering in the U.S. Domain (For Free)", by R. K. Owen, July 1997, the title is very misleading. The era of free US domain registration is over. Most of the incorporated cities of the US domain have been delegated to entities which usually charge a yearly fee for registration and maintenance.

The assignment of delegation was done by Jon Postel and done without any public notification or bidding on administrating. That is, there is no indication of why an administrator acquired the domain, but whoever they are, they have no reason to not charge any amount they wish.

Those domains not covered by incorporated cities, which are still administered by ISI direct will shortly be farmed out as well, as Postal claims ISI (Information Sciences Institute, USC) is shutting down such services.

Even worse, previously delegated domains down to one's own personal name are transferred, and those now holding such domains may be liable for the fee or lose their database references. —John Clark jclark@cts.com

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Linux in the Mainstream?

Phil Hughes

Issue #43, November 1997

Even as Linux's popularity and visibility increase, Windows NT is becoming the server of choice for corporate America.

Recently, Linux has been showing up regularly in the mainstream news. For example, there is an article titled "The Greatest OS That (N)ever Was" in the August issue of *Wired* which is both technically and historically accurate. This article will certainly help increase the visibility of Linux.

Even as Linux's popularity and visibility increase, Windows NT is becoming the server of choice for corporate America. Perhaps choice is not the right word, but NT is fast becoming the automatic answer for companies that don't want to rock the boat.

It wasn't that many years ago that everyone was saying "You can't go wrong buying IBM." At that time IBM was the 360 mainframe and its offspring. It was hard to imagine that any other company or operating system could take the place of this *big iron*. Unisys, CDC and others had pieces of the total mainframe market, but IBM seemed to be unstoppable. In fact, people commonly said "IBM machine" when referring to a computer.

There is a lot to learn from this experience. Was it good marketing on the part of IBM that made them the standard? Initially, I expect it was. After all, marketing consists of doing research, finding a need, matching your product or service to that need and advertising the benefits of your solution.

Once other companies started to offer solutions that were as good or better, IBM took advantage of its huge market share to impose itself on the consumer and squelch the competition. For example, even though a standard character set, ASCII, existed, IBM used EBCDIC, their own character set. To enter the market the competition was forced to be compatible with IBM and thus had to use EBCDIC.

As machines became more and more powerful, minis and then micros began to take over the office. At this point the game changed. The new IBM was not a hardware/software giant like IBM but the software-only company, Microsoft. Microsoft offered a better solution which, at the time, meant a more cost-effective solution. As a result, they got the market share and that is where we are today.

Microsoft has clearly overpowered their competition in the IBM tradition and has vendors such as Intel going along for the ride. The *standard* is to be Microsoft compatible; Plug-n-Play means working with Microsoft operating systems. Virtually all PC hardware vendors bundle a Microsoft operating system with the hardware.

The Linux community is playing catch-up, developing software like SAMBA so that Linux will interact with Microsoft systems, dealing with Plug-n-Play hardware that only works with Microsoft operating systems and so forth. So, we have software that addresses compatibility issues. Now, we need the hook—the thing that will make Linux a better solution. That hook isn't going to come from out-spending Microsoft, it has to come from innovation.

What Should We Do?

I am more of an observer than an expert, but I do have some suggestions. Some of these ideas come from a Seattle-based Linux mailing list, the GLUE list (check out <http://www.ssc.com/glue/>) and from talking to people about what they want.

1. We need a certification program—a way to say that someone is a certified Linux technician. Red Hat has started this effort (check out <http://www.interweft.com.au/redhat/>) with what they call “The Red Hat Commercial Support System”. I talked to Robert Hart about it, and he pointed out that since the effort is needed, Red Hat decided to go ahead on their own to get it moving. Hopefully, other vendors, or possibly Linux International, will turn this idea into an industry-wide effort.
2. We need a service business package. It is equally applicable to any business where the service provider travels to the homes of his customers—anything from a window washer to a roofing contractor. Software for these types of entrepreneurs needs to run on a laptop, which includes a printer. The issues that must be addressed are customer database, cost estimation, invoicing, inventory and travel route planning. This is not a trivial application, however, once all the pieces are integrated, it offers some great potential markets. While each specific market (Contractors, for example) is not huge, the possibilities are enormous. Also, these various industries have newsletters, discussion groups and, in some cases,

magazines. Offer an innovative solution to such an industry and word will spread quickly.

3. There are packages for FAXing and voice mail from Linux. Putting together a commercial-quality solution for a small business would make a great stand-alone application. As the majority of the interface between users and the system would be via the telephone, there is little concern for users having to learn to use a new computer. The FAX interface could be handled through an office e-mail gateway (which Linux could handle) again isolating the office worker from the actual Linux command line.
4. Another huge market is government. Governments tend to have limited budgets and want a solution that just addresses their particular problem. Also, governments tend to communicate with each other—particularly at the city and county level. This is advantageous as it means that a good solution would require little marketing beyond a couple of success stories. Government solutions could include meeting-room scheduling, general word processing, accounting, Internet connectivity and web servers. The main difference between this market and the general market is the ease with which you could market a tailor-made solution. It also offers a good chance for local consultants to get involved.

All of these solutions need a truly easy install. By this I mean a bootable CD which installs and asks for the information it needs. While it might be difficult to make a universal Linux system this easy, for a specific market, it should be easy.

Remember, we don't have the money to be bigger than Microsoft. We need to go after markets that better fit our size. If we solve the problems of a few of the smaller markets, people will start coming to us for a solution.

Let's Talk About the Ideas

We have established some HyperNews discussion groups for the various topics on the *Linux Journal* web page at <http://www.linuxjournal.com/>. Click on the "Discussion Groups" button and jump in.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Ownership of Linux Trademark Resolved

Marjorie Richardson

Issue #43, November 1997

The only thing that could have made this announcement better is if we had been able to make it back in January.

After one year of work the trademark dispute with William R. Della Croce, Jr. was finally resolved. Everyone at *Linux Journal* is pleased that this troublesome issue has come to a satisfactory close. We feel that this most happy result is due to the hard work of Mr. Jerry Davis and the overwhelming support given by the Linux Community. The only thing that could have made this announcement better is if we had been able to make it back in January.

By this time many of you have probably read the official press release. For those who have not, it is printed below.

Press Release

Monterey, California, August 20, 1997—A long standing dispute over ownership of the Linux operating system trademark has just been resolved. As a result of litigation brought by a group of five Linux companies and individuals against William R. Della Croce, Jr. of Boston, Massachusetts, Della Croce has assigned ownership for the registered mark to Linux Torvalds, the original author of Linux, as part of the a settlement agreement.

The plaintiffs in the suit were Linus Torvalds; Specialized Systems Systems Consultants, Inc. (*Linux Journal*) of Seattle; Yggdrasil computing, Inc. in San Jose; Linux International, Amherst, NH; and Work Group Solutions of Aurora, CO. Non-plaintiffs Red Hat Software, Inc., Metro Link Inc. and Digital Equipment Corporation supported the litigation and contributed to the cost of the litigation.

The five plaintiffs brought suit against Della Croce in the U.S. Trademark Trial and Appeals Board, in November 1996. Della Croce had obtained registration of

the Linux mark in September 1995, which created a storm of protests by the Linux community, who felt the mark belonged to Torvalds or the Linux community and not to any individual. In an attempt to correct the situation, the plaintiffs retained the internationally known intellectual property law firm of Davis & Schroeder of Monterey, California, who handled the case on a greatly reduced fee bases, as a service to the Linux community.

The five plaintiffs, through their attorneys, announced today that (1) the matter has been settled by the assignment of the mark to Linus Torvalds, on behalf of all Petitioners and Linux users, and the dismissal with prejudice of the pending PTO Cancellation Proceeding; and (2) that Respondent was reimbursed for his trademark filing fees and costs by Petitioners. The other terms of the Settlement Agreement are confidential.

All inquiries should be referred to Petitioners' law firm, Davis & Schroeder at 408-649-1122 or by email at ggd@iplawyers.com. A copy of the original Cancellation Petition filed in the TTAB, can be found at <http://www.iplawyers.com/text/linux.htm>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Power Printing With MagicFilter

Bill W. Cunningham

Issue #43, November 1997

Here's a program to take the work out of printing—just like magic.

Printing with Linux can be a daunting experience—it needn't be. If you really want to become a printing expert, check out the Linux Printing HOWTO, the Linux Printing Usage HOWTO and the **lp** family of man pages. However, if you just want your Linux box to print text, PostScript, graphics and other formats you throw at it today, read on.

When something sounds too good to be true, it usually is. The one exception to that rule is MagicFilter-1.2, by H. Peter Anvin. It's an extensible, customizable print filter that enables just about any kind of printer to print just about any kind of file automatically.

MagicFilter Description

MagicFilter has two jobs. First, it detects the type of file being printed. It does this by searching for key strings at specific offsets in the file. It does not rely on file names or extensions as indicators of file type. For instance, if the file's first two characters are %!, it's processed as a PostScript file. Other file types, such as gifs, bmps, etc., are detected by similar rules. MagicFilter's second job is to call an appropriate conversion utility to print the file on your specific printer. If you happen to try to print something unprintable, like a binary file, the job is aborted, and you'll be advised (via e-mail) that your file couldn't be printed.

Benefit

The obvious benefit is that with MagicFilter installed, you can enter the command:

```
lpr first.txt second.ps third.gif fourth.jpeg\  
  README.gz
```

at the prompt, and all five files will spool up and print in fine style. Another perhaps greater benefit is that most packages for Linux come preconfigured with `lp` or `lpr` as the default printer. With MagicFilter installed, all of your favorite programs like Pine, Netscape, Applixware, etc. should print fine right out of the box. I'll go into more detail on printing with applications later.

MagicFilter Installation

Before you can build MagicFilter, you're going to need some supporting packages. At a minimum, you need the **gcc** compiler. MagicFilter's Makefile is generated by a script called **configure**, that scans the directories in your path and caches the locations of any conversion utilities that it finds. If `configure` doesn't find some of the utilities, MagicFilter will build anyway, but without the ability to print some file types. Table 1 shows the support packages needed for varying levels of file printing ability.

Table 1

The staircase effect mentioned in Table 1 occurs when a printer omits the carriage return from end-of-line sequences. If you want to see the staircase effect in action, send a short text file directly to your printer device by running this example (substituting your `lp` device name):

```
cp /etc/group /dev/lp1
```

Decide what level of robustness you need in MagicFilter, install those supporting packages, then build **magicfilter**. I have listed some sources for these packages at the end of this article.

All right, let's assume you have all your support packages identified and installed somewhere in your path. You've obtained the file `magicfilter-1.2.tar.gz` and as root, moved it to a suitable location. (I always use `/usr/local`.) Unpack the file with this command:

```
tar -xzf magicfilter-1.2.tar.gz
```

This command unpacks MagicFilter into the newly-created directory `/usr/local/magicfilter-1.2`. Change directory to `magicfilter-1.2`, and run the `configure` script. Note the output displayed on your screen; you will probably see a few packages that `configure` didn't find. Peter Anvin assures me that if you have all the packages shown in Table 1 installed and in your path, you will build a fully-functional `magicfilter` program. Some of the packages searched for have been replaced (in functionality, at least) by newer ones in the support packages. Anyway, now run **make**; next, run **make<!s>install**. Afterwards, the directory `/usr/local/magicfilter-1.2/filters` will contain about 40 scripts for just about any

printer you are likely to encounter. The actual magicfilter binary may be in /usr/local/bin. What you do at this point is copy or move the script(s) for your printer(s) to /usr/local/bin. For example, if you have a Canon BJ-200ex printer, your script is bj200-filter. Some printers may not have a script per se, but there should be a script that works for almost any printer.

Optionally, you can run **make<\ls>install_filters**, and all the scripts will be copied to /usr/local/bin. It's worth noting at this point that /usr/local/bin (or wherever you've installed MagicFilter) does not need to be in your path. When we reference these files next, we will use absolute path names.

Since I'm writing about Linux, which uses the BSD printing system, I will only address the Linux-specific installation in the /etc/printcap file. Be advised that not all flavors of Unix implement this printing system. Edit the /etc/printcap file and weave in a call to your printer's script as an input filter. Here is an example for a system with two printers, a HP LaserJet 4 and a HP DeskJet 550C on /dev/lp1 and /dev/lp2, respectively:

```
pencil|lp|PostScript|ljet4|HP LaserJet 4:\
:lp=/dev/lp1:sd=/var/spool/lpd/pencil:sh:mx#0:\
:if=/usr/local/bin/ljet4-filter:
crayon|dj550c|color|HP DeskJet 550C:\
:lp=/dev/lp2:sd=/var/spool/lpd/crayon:sh:mx#0:\
:if=/usr/local/bin/dj550c-filter:
```

The syntax for the /etc/printcap file is a subject in itself. See the Linux Printing-HOWTO a complete description. Peter points out, "Note the alias `lp' for the default (text) printer, and `PostScript' for the preferred PostScript printer."

Next, kill and restart your **lpd** (print demon). Mine killed okay, but it wouldn't restart properly; so, I rebooted. Following the reboot, lpd was indeed running again.

Power Demonstration

Now it's time for some tests. First, try printing a normal text file from your prompt; something short, like:

```
lpr /etc/group
```

Your printer should print the file without the staircase effect. Assuming no problems here, try printing a PostScript (.ps) file. Next, try a graphics file like a .gif or .jpeg. If you have one handy, try printing a .dvi file. All these files should produce perfect output. Finally, try printing a binary file like /bin/sync. You should *not* get any output from your printer, and, if you're running **sendmail**, **smail** or other mail agent, you should get an almost immediate e-mail from bin@*your.localhost* explaining why your file could not be printed. If you're not

running a mailer, your \$ prompt will return, and you will get no output from your printer. MagicFilter does not display error messages on the console.

At this point, it's worthwhile to print a listing of the actual filter scripts for your printer. You'll notice that it's an interpreted script with MagicFilter as the interpreter. It's laid out in three vertical columns. Column one is the offset from the file's beginning, column two is the string of characters searched for and column three is the facility for dealing with each type of file. Reading down the column, you can see all the different types of files that MagicFilter can handle and how it handles them, as well as the ones it can't. Note the **pipe** and **fpipe** methods in column three. That method's output is fed back into MagicFilter for a second or subsequent pass. This is necessary in the case of .dvi files, for example. First **dvips** runs on the .dvi file, and the resulting temporary PostScript file is fed back into MagicFilter where **ghostscript** then performs the actual printing—pretty slick.

Pine

This popular e-mailer requires a little tweaking to work properly. If you leave Pine's default printing mechanism set to `lpr`, you may find that e-mails of more than one page will lose the last 2 or 3 lines of text on some printers. This is due to the fact that some printers, Canon ink jets for one, can't print more than 63 lines per page, and `lpr` thinks they ought to have 66 lines per page. To fix this, create a short script called **print**, containing these two lines:

```
#!/bin/sh
pr $1 | lpr
```

Put this script in your home directory, or somewhere in your path, then install it as Pine's printing command. From Pine's main menu, select as follows: S (setup), P (printers), then arrow down to "Personally selected print command". Set "Printer List" to `/dev/lp1` (or whatever your printer device is named) and your print command (on that same line) to `~/print`. To make these changes, enter **C-N** (change, name) and then **C** again (change command). Save your changes when prompted.

This printing setup will run your e-mail through the **pr** utility for formatting before spooling to print. This is kind of neat, because your pages of output will have a date/time stamp and page number on the top of each page. You can modify `pr`'s behavior with command line switches to produce the appearance you wish (see `man(1) pr`).

Netscape

You should be able to print from Netscape without any tweaks at all. To test it, fire up Netscape and load a challenging document. From the File menu, select

Print and observe the dialog that opens up. From the top of the dialog, verify that the "Printer" button is pushed, that your "Print Command" is lpr and that the rest of the print settings make sense for your printer and paper. Then, press the "Print" button. Figure 1 shows how this should look.

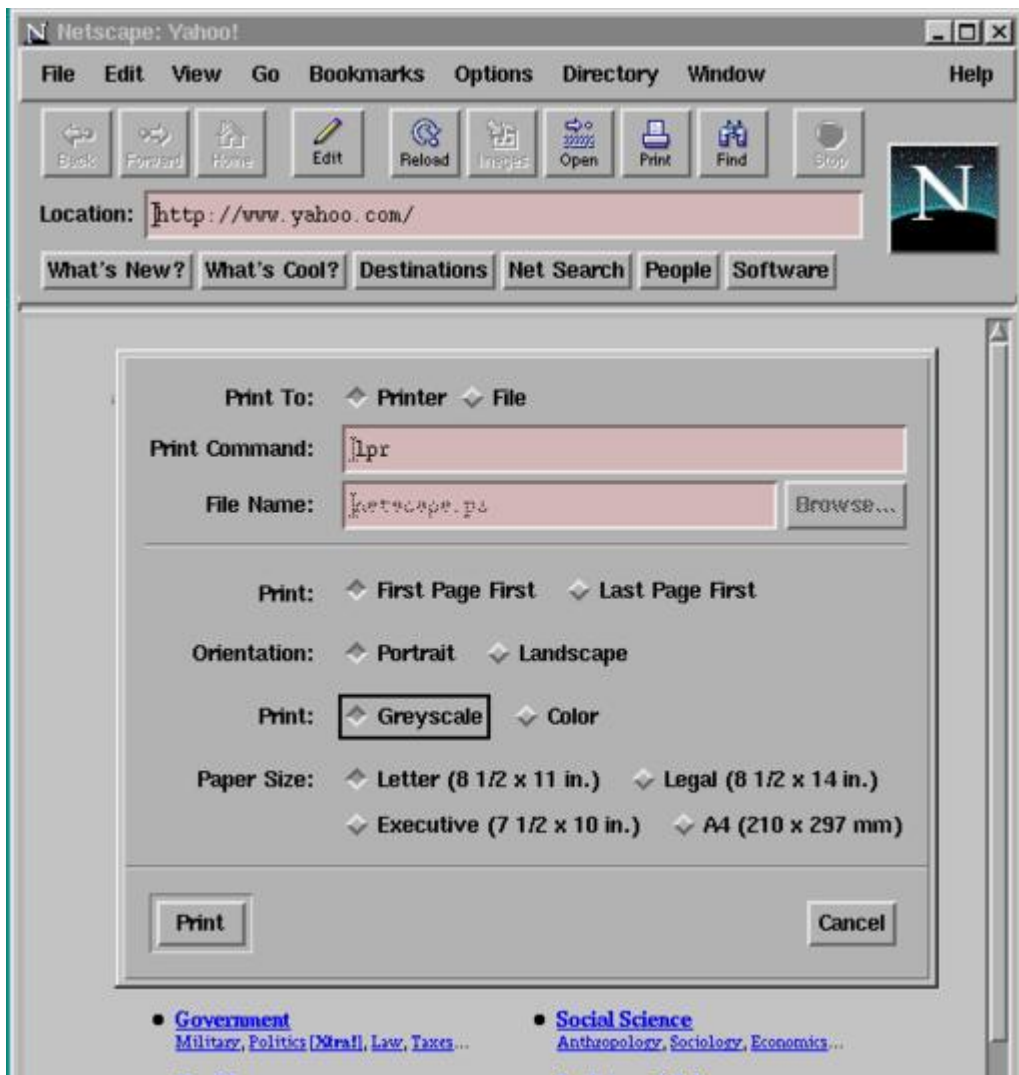


Figure 1. Netscape Dialog

You should get a printout of exactly what's on your browser screen, graphics and all. Of course, this assumes you have a graphics-capable printer and that you built MagicFilter with at least ghostscript support. Netscape defaults are stored in a file called `~/Netscape/preferences`. You can check that file to make sure your print command is `lpr` and adjust it, if necessary.

You can even use Netscape as a sort of kludgy word processor. From the File Menu, select "New Document -> Blank". The Netscape Editor will open up with a blank screen, ready to copy. This editor is geared toward HTML; it's great for writing WWW pages, but for producing letters, etc., it leaves a lot to be desired. With Netscape's editor, what you see on the screen usually does *not* exactly match the printer output.

Applixware

For real, actual word processing, I don't think you can beat Applixware, especially since its price just dropped to \$199US a copy. As a matter of fact, I'm typing this article on the Applixware word processor. It's only officially supported on Red Hat Linux, but it works fine on Slackware 96, too. When installing, be sure to set your DISPLAY environment variable to:

```
export DISPLAY=:0
```

On a 486 platform, commenting out Speedo and Type I fonts in the /etc/XF86Config file allows Applixware to load fast enough to prevent the X server connection from timing out. This tweak makes Netscape load much faster, too. Also on a 486, Applixware works in an X-only environment, i.e., restart /sbin/init at level 4 instead of the usual level 3. Finally, to run Applixware in general, have a whole lot of RAM.

To print a project with Applixware, push the print icon on the top menu bar and note the print dialog that follows. Just press on lp under "Printers", set "Class" to PostScript (should already be there), make sure the "Print to File" button is not pressed and click OK. With MagicFilter installed, Applixware should literally print right out of the box. Figure 2 shows how the print dialog appears just before printing.

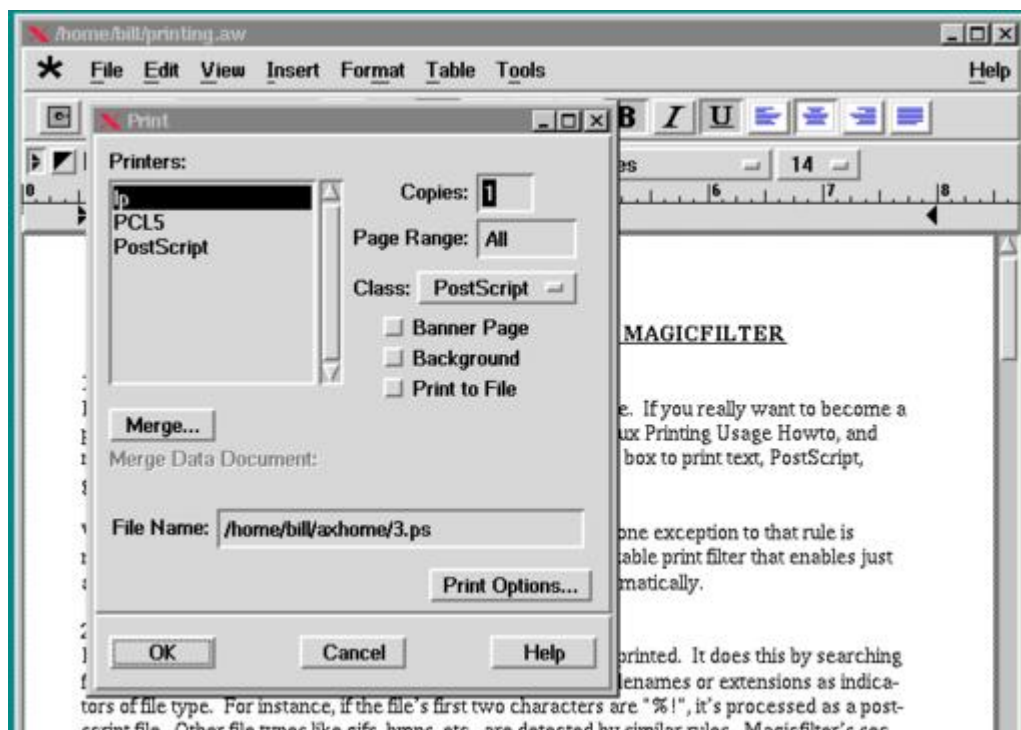


Figure 2. Applixware Dialog

TeX/LaTeX

Ordinarily, printing a .tex file is a three-step procedure. First, you run **tex** on the .tex file, then dvips on the .dvi file and then ghostscript on the .ps file. With MagicFilter, you just run TeX or LaTeX on your document file, shoot the resulting .dvi file straight to lpr and let MagicFilter handle the rest. One caveat here—if the .dvi file contains references to PostScript figures, you must run dvips manually and send the generated .ps file to lpr. Since the default for dvips is to send the output to the printer, you can simply think of it as running dvips to send the document to the printer. This is not a MagicFilter flaw, but rather it's a “miss-feature” in the way .dvi files handle embedded PostScript.

Conclusion

After using MagicFilter for a while, I thought it would make a great addition to commercial Linux packages as part of their initial installation. But on second thought, this would be easier said than done, given the many kinds of printers and file formats we have today. One of MagicFilter's strong points is the fact that it is tailored to your personal system, printer and printing needs. To try to integrate it into a distribution's installation process might turn out to be counterproductive. MagicFilter and its supporting packages are easy to build, and they're readily available. It's probably best to leave things the way they are. If you've never compiled software on your own system, MagicFilter makes a great first project and could be a big confidence builder for a new Linux user.

Brian McCauley makes the point in the Linux Printing-HOWTO that MagicFilter prevents the user from printing a listing of, for example, a binary file, and he's right. However, I think most people would prefer to turn off MagicFilter, if this situation ever came up, print the listing and turn MagicFilter back on. It just makes life so much simpler.

Software Sources

MagicFilter's Author



Gunnery Sergeant **Bill W. Cunningham**, U. S. Marine Corps, is the Network Administrator at Marine Forces South, Panama Canal Zone. He, his lovely wife, and four wonderful kids will return to the United States this fall. He enjoys

running, biking, playing Tetris late at night and plans to develop children's software for Linux in his soon-to-commence second career.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

ssh: Secure Shell

Alessandro Rubini

Issue #43, November 1997

Mr. Rubini gives us an introduction to the ssh program suite and its features, so that we can establish secure communication channels.

The **ssh** program suite is an interesting way to establish a secure communication channel between hosts across the Internet. It replaces a few usual communication channels (such as **telnet**, **rlogin**, **rsh** and **rcp**) and provides equivalent functionality via a new secure protocol. This article is meant to introduce the user to ssh features without any pretense of being an authoritative tutorial on security issues.

Bureaucracy

First of all, as usual with encryption material, let's throw in some legalese. The Unix implementation of the package is distributed free of charge for non-commercial use and can be retrieved from major ftp sites around the world. The meaning of "non-commercial use" is well stated in the **COPYING** file within the ssh distribution. Non-commercial use includes use of the tool by people who work on the Internet, provided ssh is not used as a selling argument. However, being allowed by the license is not enough to use the package. You must also ensure your government allows using cryptographic material: some countries prohibit citizens from using any type of cryptography.

As far as standardization is concerned, the IETF (Internet Engineering Task Force) is working on the ssh idea. This means that the protocol is going to be standardized and will always remain thoroughly documented. Anyone can reimplement the software and distribute her version with different copying policies. Although there is no RFC about ssh yet, three Internet drafts are already available which document the different aspects of the protocol. The drafts I downloaded in June are going to expire on September 1, 1997, but new drafts (or a real Request For Comments) will be released before that time.

The Problem

The main aim of ssh is overcoming the security problems of TELNET and rsh-like protocols. These protocols, though widely used, are very weak as far as security is concerned: TELNET transmits clear-text passwords over the network, while rsh and friends are easily broken by spoofing techniques (skilled system administrators or anyone with physical access to the wire can exploit the .rhosts files to be granted access on remote computers). These vulnerabilities are addressed by the ssh protocol by enforcing a strong authentication scheme between the communicating parties.

The protocol not only authenticates the client host with the server, but also enforces encryption of all the data sent through the established communication channel. Although encryption requires some computational load in the communicating hosts, the user can be confident that no sniffer can make sense out of network packets, and nobody can forge packets to be granted access with the server.

Finally, the protocol is designed to allow encapsulation of other communication channels within an ssh stream, so that non-encrypted protocols can benefit from ssh capabilities at no cost. This feature is most useful for establishing secure X11 connections, but its use is much more flexible than just that security.

As far as the user interface is concerned, the new programs are designed as drop-in replacements for **rsh**, **rlogin** and **rcp**. While the system administrator needs to deal with the ssh issue at least to install the package, the final user can happily ignore the fact that the underlying communication mechanisms have been changed.

Overview of the Package

The current version of ssh, as I'm writing this article, is 1.2.20. It is distributed with a **configure** script, so that:

```
./configure && make && make install
```

does the complete job of installing the package. Even if **/usr/local** belongs to you, you'll need to be a privileged user in order to install ssh. This requirement is necessary, because both the server and some client programs must be able to open privileged TCP ports. Moreover, the package installs its system-wide files in the **/etc** directory. If you must run an ssh session without installing the programs as a privileged user, you should first read the FAQ file, which describes how to accomplish this without too much trouble.

The program suite is made up of a server program, a few clients and a pair of support tools. In order to become an ssh server, the suggested action is to invoke **sshd** from your initialization files when the computer boots. The program is installed by default in the `/usr/local/sbin` directory, and it listens to port 22/tcp of the local host to accept incoming connections. You should add a line like the following one to the `/etc/services` file, if it is not already included:

```
ssh      22/tcp  # secure shell
```

It is possible to run `sshd` from `/etc/inetd.conf`, by passing the `-i` flag to the daemon. This practice is discouraged, because the daemon can take several seconds to start, as it generates an encryption key each time it starts.

On the client side, nothing needs to be done at installation time to enable use of the services. After installation, several programs can be found in the `/usr/local/bin` directory that can be executed by anyone. Here's a list of these programs:

- **ssh** (slogin): The ssh client program behaves like rsh in that it executes a command on a remote host, redirecting its own stdin, stdout and stderr to the remote. If only one argument is specified in the command line, it is taken as the remote host, and ssh opens a terminal session on that host, just as **rlogin** does. The **slogin** client is just a symbolic link to ssh. Usually, the user is asked to provide a password for authentication on the remote host.
- **scp**: "Secure CoPy" is a replacement for "Remote CoPy". It uses the same syntax and performs the same task, with the same security enhancements offered by ssh. The user is usually prompted for a password to perform authentication.
- **ssh-keygen**: This program generates new keys associated to the user who invokes it. The public key and the private key generated are saved by default in the directory `$HOME/.ssh/`, in two files called `identity` and `identity.pub`. A pair of keys is needed, when you wish to securely log into the local host from outside without providing a password.
- **ssh-agent**: This program is meant to manage security for a user session. It stores the private keys for the user and can be contacted by any child process. See the man page for more information about ssh-agent.
- **ssh-add**: This program adds identities to the authentication agent. **ssh-agent** must be an ancestor of the process running ssh-add in order for them to communicate.
- **ssh-askpass**: This short program is used internally by ssh-add and ssh-keygen to ask for a pass phrase using the X graphical environment.
- **make-ssh-known-hosts**: This Perl script retrieves the public keys of all hosts in a domain; it queries the DNS and talks with hosts belonging to a

domain. This program updates the `/etc/ssh_known_hosts` file to hold the new keys. The program should be run periodically, usually via **cron**, to reflect any change in host public keys or new installations of ssh on hosts in the local networks.

Although the number of installed files might scare a novice user, there's nothing new to learn as long as you just use the plain ssh and scp client programs. The system administrator can even choose to replace ssh for rsh and scp for rcp, so that secure operation is completely transparent to the final user. Actually, such a replacement is not completely transparent, since ssh introduces a new feature not present in plain rsh or rlogin: X11 forwarding. This feature is considered a "side effect" of using the ssh protocol and is described in the next section.

The role of the other programs is secondary. They help to set up a good working environment to avoid typing too many passwords during normal operation without compromising system security. While knowing their internals is not essential for program operation, a quick look over the documentation might provide interesting insights about current encryption technology and can help in configuring hosts to automate the login process.

X11 Forwarding

The ssh protocol is designed to be flexible and supports multiplexing of several communication channels within a single TCP stream. This design choice results in two effects: on one side the implementation of the protocol is much more elaborate than other TCP-based protocols, and on the other side the final user can exploit the added flexibility to achieve new goals. One of these goals is establishing secure communication channels between the X server and client applications. This feature is enabled by default whenever an ssh session is established.

The idea behind X11 forwarding is quite straightforward. The ssh application runs locally and is able to connect to the local X server without resorting to the network (via local Unix-domain sockets). Remote graphic programs, on the other hand, can connect locally to the sshd server which spawned the remote shell (via the loopback network interface). The remote sshd can encapsulate graphic data in the secure communication channel it owns, to complete the path linking the graphic application and the X server.

Figure 1 shows how a remote X application (running on a computer named *sandra*) securely connects to the local X server (on *morgana*).

Figure 1. Secure Connection to X Server

When you log into a remote computer through ssh, the DISPLAY environment variable is automatically set to a proper value, and no user intervention is needed to establish the graphical channel. The following screen-shot shows automatic assignment of DISPLAY:

```
morgana% ssh sandra env | grep DISPLAY
DISPLAY=sandra.systemy.it:10.0
```

It is apparent how any graphic program invoked on sandra by the ssh session will connect to a local display (i.e., sandra:10).

The ssh/sshd programs can also forward other TCP channels, according to the user's needs. This capability can be activated by specifying command-line switches to the client ssh program. I won't describe the mechanisms here, as the manual page for ssh is well written.

Authentication

The main problem when establishing a connection through an insecure network is performing reliable authentication. The ssh package is quite pedantic about authentication, and you'll be prompted for your password more frequently than usual. Typing passwords over and over is distressing and can be avoided by proper configuration of system files. Note also that any password you type is transmitted *after* establishing the encrypted communication channel.

You can type **ssh -v** (verbose) to get a report on what is happening. The information returned is very useful if you are unexpectedly prompted for a password. Now, let's look at the steps performed by ssh to authenticate a user in the remote server.

First, if the target account has no password, access is granted. If it does, different kinds of authentication engines are tried; each of which can be enabled or disabled into the server. For example, by default "PasswordAuthentication" and "RhostsRSAAuthentication" are enabled, and "RhostsAuthentication" is disabled.

The following is the sequence of actions when you try to log into a server running with the default configuration—which can be changed in `/etc/sshd_config`.

1. The client receives the public key of the server. If it is not recognized, ssh asks the user interactively if the connection must be continued. By confirming, the user trusts that the remote host matches its name, and the public key of the server is saved on the client, in the file `$HOME/.ssh/`

known_hosts. This step is not performed if the server host is known system-wide (i.e., it appears in /etc/ssh_known_hosts).

2. The client tries authenticating through "RhostsRSA". This requires that "Rhosts" authentication succeeds: either .rhosts in the user's home directory or /etc/hosts.equiv allows login. **sshd** is more pedantic than **rlogind** in checking these files and denies permission if any of the files are group-writable or world-writable. Needless to say, the entry beginning with the "plus" character in either file is disregarded. Moreover, .rhosts is not even used if the home directory of the user is group-writable or world-writable, and /etc/hosts.equiv is not used to authorize root logins. In addition to the standard files, sshd also checks .shosts in the home directory of the user and /etc/shosts.equiv. These files are useful if you still wish to run rshd or rlogind on the server hosts by trusting fewer hosts than you trust via ssh.
3. If the previous step succeeds, RSA is tried (Random-State Authentication). This technique consists in the client sending a challenge to the server, which must reply correctly. The challenge consists of random data encrypted using the client's private key; the server must decrypt such data and return its checksum. The server can solve the challenge only if it knows the public key of the client, which is known only if the remote user agreed to trust the client (local) host. RSA is used to prevent authorizing untrusted hosts which forge DNS records or which temporarily steal the IP address of a trusted host.
4. If either of the previous steps fails, i.e., if "RhostsRSA Authentication" as a whole fails, the client reverts to "Password Authentication", by asking for a password from the local user.

If your .rhosts file is correctly configured and you are still prompted for a password, the problem is most likely caused by RSA not succeeding. The easiest way to store the client's public key in the server is invoking ssh right away to connect back to the client computer. When confirming to continue the connection, the server (now acting as a client) downloads the public key of the local host (now acting as a server).

Algorithms and Future Directions.

The design of ssh is full of hooks for future extensibility. First of all, the client and the server exchange a "software version" and a "protocol version" at the beginning of each section. While the "software version" is mainly used in debugging problems, the "protocol version" is a great resource to accomplish smooth upgrading from one version of the software to the next one. Both the client and the server are required to support at least the previous version of the protocol, in addition to the current one. This requirement is designed to help deal with the transition period whenever the protocol gets enhanced

(which doesn't happen too often). When running `ssh -v`, you can see, among other things, the exchange of version strings.

Another great design feature of the protocol is that new cryptographic algorithms ("ciphers") can be added to the basic machinery without losing generality. This is accomplished by making the choice of the cipher to use at runtime. During handshake (the first few packets being exchanged by the communicating parties), the server declares which ciphers it supports, and the client chooses one of those ciphers. Every ssh implementation is required to support at least 3DES, in order to ensure a secure link can be established between any client and any server. Users and/or organizations are, nevertheless, free to implement new ciphers and specify them as the default choice. A few ciphers are part of the official ssh distribution, and the user can ask for a specific algorithm on the ssh command line to override the default.

The protocol also supports compression of session data. A compressed session can actually be faster than a non-compressed one, if the local network is slightly loaded. Once again, compression is optional, and the communicating parties agree whether or not to use it.

The standardization efforts endorsed by the IETF are aimed at defining version 2.0 of the secure shell protocol (the version supported by ssh-1.2.20 is called 1.5). The Internet drafts currently available document three different aspects of the upcoming 2.0 protocol:

1. the connection protocol, draft-ietf-secsh-connect-00.txt),
2. the transport-layer protocol, draft-ietf-secsh-transport-00.txt)
3. the authentication protocol, draft-ietf-secsh-userauth-00.txt.

These documents are quite technical, but very interesting to peruse. The protocol the IETF is working on looks promising, giving even more flexibility than the current one.

The curious reader is urged to browse the network to retrieve more information on these topics. I can provide a few pointers to begin with, but I'm pretty sure you'll find several more pointers about this kind of topic.

Resources



Alessandro (rubini@linux.it) is a member of the “Pluto” Italian user group, which is going to meet in Perugia, Italy, during November. See <http://www.pluto.linux.it/> for details.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Highway POS System

Marc L. Allen

Issue #43, November 1997

Why one petroleum company decided to use Linux for its new point of sale computers.

Why Linux? Because it works. Because it does the job. Because, let's face it, it's free.

In late 1993, we started doing preliminary design work on a new retail petroleum point of sale (POS) unit. Basically, it's an electronic cash register (ECR) which also handles credit cards and controls fuel dispenser systems. We wanted to design a PC-based system to leverage the ease of availability and low prices associated with PC hardware.

Our sister division in the Netherlands had already begun designing a new POS system for the European market. Since they had provided major components of our current successful products, we waited in heady anticipation for their design documents. Unfortunately, their design was based on MU-DOS, a multitasking DOS by Digital Research. We weren't very happy, and started researching alternative operating systems.

Requirements

We had the following requirements:

- Soft real time: While we didn't need a true real time system, we did need an operating system which would be able to handle over a dozen serial ports. These ports weren't always active, but we'd have bursts of communications from 9600 baud and higher, and we needed to be responsive in processing the incoming data. We also had to handle an experienced ECR cashier who would be entering keystrokes at a high rate and expecting a quick system response.

- Multitasking: Preferably preemptive. What we really wanted was protected memory segments, so that one task wouldn't be able to corrupt another's data space. Being able to use multiple programs would allow us to change out specific programs without affecting the rest of the system.
- Non-proprietary: We wanted an OS and development system that was supported by an outside agency using industry standard tools and utilities. We wanted to make our own workload as small as possible.
- Inexpensive: The price point we were aiming at for the product didn't allow too much choice in the way of software.

Other Operating Systems

In fairly short order, we had examined a number of candidates and decided to use SCO Unix w/Chorus Micro-kernel. Linux hadn't become an option—yet. We had heard of it, but didn't feel it was mature enough or commercially acceptable enough to warrant serious investigation.

Windows 3.1 still seemed too buggy, Windows NT was too expensive and too much of a resource hog.

QNX suffered from a lack of virtual memory. While we didn't have a real need for virtual memory, we wanted the ability to use it if the application needed it for some short but highly memory intensive operation.

OS/2 was originally high on our list, as our sister division was also looking at migrating to OS/2 for their product. Unfortunately, we consistently had problems with installation and found support extremely difficult to get.

Deciding on Linux

SCO's only real problem seemed to be pricing. They wanted much too much, much too soon. While we started negotiating, we needed to start developing. We didn't want to purchase any SCO development systems, so we decided to start our development on Linux. Linux was fairly POSIX compliant, so we were confident we would be able to port our code without difficulty to another platform, particularly another Unix platform.

Differences in usability showed up almost immediately. I was working on the one SCO system we did have (an evaluation unit) and found the utility commands to be almost primitive compared to Linux. Things as simple as the recursive (-R) flag were missing from the **chmod** and **chown** commands. I found myself locating and acquiring more and more GNU tools. I was unable to get good or timely support for the SCO C compiler and the C++ front end. (They didn't have a native C++ compiler at the time.) I had difficulty in getting quality technical support. I had plenty of their time, but kept talking to support

personnel who were not close enough to the module for which I needed information. I continually had to bother my SCO sales representative to renew my support contract (which was free due to the on-going negotiations).

On the Linux side of the partition, life was much smoother. The utilities were very advanced, the documentation reasonably good and the Linux newsgroups were a wealth of information and solutions. We tried using the SCO newsgroups, but found that, unless someone had experienced the exact same problem, little help was forthcoming. Linux newsgroups allowed us to contact the actual authors and maintainers of specific areas of code to better understand and troubleshoot problems. They were also fast. Within a matter of a day or two (sometimes less) we had fixes, workarounds and suggestions to try. It was almost like having a bunch of Linux experts on our payroll.

While upper management continued negotiating with SCO, engineering started quietly hoping that we could stay with Linux. However, we still had that commercial issue. We were afraid our customers wouldn't accept a free "hackers" OS. Luckily, Linux just kept gaining momentum. Linux 1.0 came out during this time, and suddenly companies like Caldera were making this wonderful OS a commercial product.

Suddenly, using Linux wasn't quite the issue it had been the previous year. Our customers still weren't sure, but their technical people started giving grudging acknowledgement that Linux just might be okay in a product. We decided to stay with Linux and drop SCO, who still wanted too much, too soon—we've never looked back.

Today

Currently, we've got Linux 2.0, demand loadable kernel modules and connectivity to Windows, Novell and AppleTalk. Most importantly, we never hear the words, "Hmmm, it looks like an OS problem." We still have the occasional problem with an application or utility, but it's rarely anything critical, and, if it is, we always have the source code and access to the author.

We aren't the only retail petroleum POS vendor to come up with a PC-based solution. We are, however, the only Unix-based one. There have been attempts based on Windows 95, OS/2 and even DOS. So far, no one's really succeeded. Our product will hit the streets around mid-year (1997). It's already done wildly better than expected in its first field trials. We have confidence that it's going to be a winner, due in large part to Linux.



Marc L. Allen works for Schlumberger Technologies, designing top-of-the-line POS systems for the retail petroleum industry. Using Linux on the job gives him enough trouble-free time to support his Duplicate Bridge habit without upsetting his wife. He can be reached at allen@chesapeake.rps.slb.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

New Products

Amy Kukuk

Issue #43, November 1997

NetTracker 3.0, ScriptEase: Integration SDK, JDesignerProTM 2.1 and more.

NetTracker 3.0

Sane Solutions, LLC has released NetTracker 3.0, a tracking program that can be installed on a web server and accessed over the Internet. The program can export data, graph and summary reports with drill-down capabilities. Users can also update NetTracker reports. NetTracker 3.0 starts at \$295 US for a single user.

Contact: Sane Solutions, LLC, 3 Preston Drive, Wickford, RI 02852, Phone: 401-295-4809, Fax: 401-295-4154, E-mail: info@sane.com, URL: <http://www.sane.com/>.

ScriptEase: Integration SDK

Nombas has developed a JavaScript-compatible interpreter called ScriptEase: Integration SDK that lets you format output, automate tasks and make system calls. The program ships with sample applications and printed documentation. Contact the company for pricing and other details.

Contact: Nombas, 64 Salem Street, Medford, MA 02155, Phone: 617-391-6595, Fax: 617-391-3842, E-mail: nombas@monbas.com, URL: <http://www.nombas.com/>.

JDesignerProTM 2.1

BulletProof Corporation has released JDesignerProTM 2.1, a Java RAD tool with a new visual system for building with JavaBeans. JDesignerPro 2.1 introduces the Instance Manager and Interaction Manager, which let developers import and develop with third-party JavaBeans components. JDesignerPro 2.1 is available for free download at the company's site.

Contact: BulletProof Corporation, 15732 Los Gatos Blvd. #525, Los Gatos, CA, 95032, Phone: 800-505-0105, E-mail: support@bulletproof.com, URL: <http://www.bulletproof.com/>.

VirtuFlex 2.0

VirtuFlex Software Corporation has released VirtuFlex 2.0. The product includes the JDBC Server and includes new features such as web-based administration screens, debugging utilities and new functions for processing web forms. The price for VirtuaFlex 2.0 begins at \$495 US per domain name and yearly subscription services are also available.

Contact: VirtuFlex Software Corporation, 930 Massachusetts Avenue, Cambridge, MA 02139, Phone: 617-497-8006, Fax: 617-492-0486, E-mail: comments@virtuflex.com, URL: <http://www.virtuflex.com/>.

PGP Version 5.0 for Personal Privacy

Pretty Good Privacy, Inc. has introduced a Linux beta version of PGP, Version 5.0 for Personal Privacy. It is available in binary format on the Intel ELF Linux architecture and can be downloaded from <http://www.pgp.com/>. Version 5.0 has a new PGP code base and features new algorithms.

Contact: Pretty Good Privacy, Inc., 2121 El Camino Real Suite 902, San Mateo, CA, 94403, Phone: 415-524-6229, Fax: 415-572-1932, E-mail: support@pgp.com, URL: <http://www.pgp.com/>.

TeraSpell 97

Teragram Corporation has announced TeraSpell 97 for Emacs, a spell checker for Emacs which incorporates visual highlights for misspelled words and on-the-fly spell checking. Visual highlights enable you to identify all spelling errors in your documents at once and ignore non-words that you may have used in your document. When you click on a misspelled word, the list of suggestions appears. TeraSpell recognizes several modes including; text, TieX, LaTeX and HTML. Free evaluation downloads, on-line ordering and delivery are available at the company's web site.

Contact: Teragram Corporation, 236 Huntington Avenue, Boston, MA 02115-4701, Phone 617-369-0100, FAX: 617- 369-0101, E-mail: info@teragram.com, URL: <http://www.teragram.com/>.

FileDrive File Transfer Server

Differential, Inc. has announced the release of the FileDrive line of secure file transfer servers. FileDrive provides secure uploads and downloads of very large

files and is compatible with existing FTP clients. The FileDrive server for Linux features a web-based remote administration system of its access control system, real-time monitor and other features.

Contact: Differential, 10054 Pasadena Ave, Cupertino, CA 95014 Phone: 408-864-0610. E-mail: info@filedrive.com, URL: <http://www.filedrive.com/>.

journyx WebTime

journyx has announced journyx WebTime, new web-based time tracking product. WebTime gathers and reports on employee time by project, accounting code and life-cycle state with a web-based GUI. Three levels of password protected authority exist for employees, managers and administrators. A free demo and a 60-day version are available at the company's web site.

Contact: journyx, 6716 Deauford Drive, Austin TX 78750, Phone: 512-345-8282, E-mail: info@journyx.com, URL: <http://www.journyx.com/>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

IP Masquerading Code Follow-up

Chris Kostick

Issue #43, November 1997

This is a follow-up article to the author's "IP Masquerading with Linux" in *Linux Journal* Issue 27.

Over a year ago this magazine graciously allowed me to write a couple of articles for them. One of them concerned the topic of IP Masquerading (July 1996, Issue 27) as supported by the Linux kernel. Since that time a number of changes have occurred during the ongoing development of Linux. I'd like to bring the *Linux Journal* readers up to date on what changes have taken place, explain a few of the technical details and take some guesses at where things are going in the future. Many of the technical details of functionality will not be revisited because overall the functionality hasn't changed. The previous article still applies in that area.

Recap

IP Masquerading is a way of performing address hiding. It may be that a company does not have enough registered IP addresses to connect all of its computers to the Internet or, if you're like me, you have one address through a local dial-up account, but three computers. Each of these circumstances can be solved by using masquerading to allow the "internal" computers access via the one external connection point. The external connection point will use masquerading to hide addresses. Figure 1 is a diagram of the local network I have as my setup.

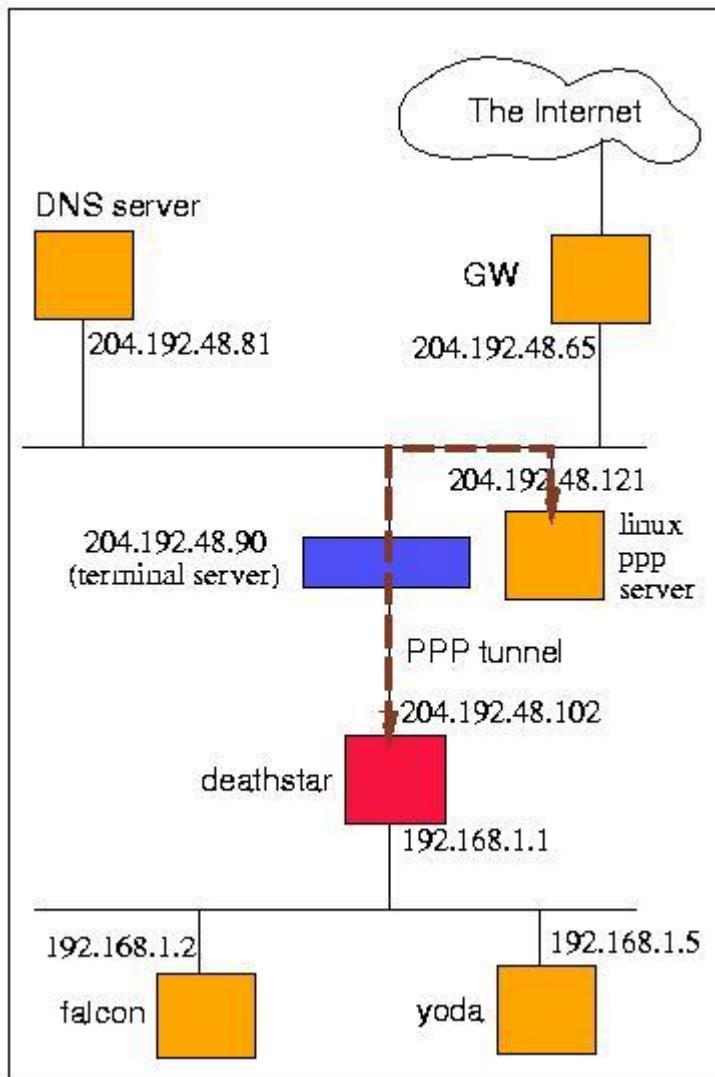


Figure 1. Masqueraded Network Diagram

Using masquerading is still as easy as ever. I have one network where I want all of my hosts to have access to the Internet. Since they are on 192.168.1.0 (the network), I can masquerade the entire class C address space. You'll notice, being the good administrator that I am, I'm using a private address as defined by RFC 1918.

To start masquerading, I defined an rc.masq file in my /etc/rc.d/ directory and added it to execute in /etc/rc.d/rc.local. The rc.masq file looks like this:

```
#!/bin/sh
#
PATH=/usr/local/bin; export PATH
```

```
#
#      setup system forwarding policy
ipfwadm -F -p deny
#
#      masquerading rules
ipfwadm -F a m -S 192.168.1.0/24 -D 0.0.0.0/0
#
#      list out the current ruleset
ipfwadm -F -l -n,
```

First, you'll notice the command to set up masquerading is the **ipfwadm** (version 2.3.0) command. This is noticeably different from the previous article when only the **ipfw** command could be used. Since kernels 1.3.66 **ipfwadm** is the only command to manipulate masquerading rules and is available from <http://www.xos.nl/linux/ipfwadm/>.

There are really just two statements given. The first one:

```
ipfwadm -F -p deny
```

defines the forwarding policy (**-F**) of this machine—deathstar from Figure 1. It sets the policy to deny all packets to be forwarded by deathstar. Forwarding is the situation where a packet has a source address and destination address different from any of deathstar's interfaces and is to be routed through.

The second statement:

```
ipfwadm -F -a m -S 192.168.1.0/24 -D 0.0.0.0/0
```

added a masquerading rule for the source (**-S**) network 192.168.1.0. We can tell it is for the entire network by the 24-bit netmask associated with it. 24 bits equates to 255.255.255.0 to indicate the network versus the host portion of an IP address. The destination (**-D**) address is the all encompassing 0.0.0.0 network, which is used to identify any network. That's it.

The last command allows me to look at the rules set for deathstar. The output looks like this:

```
IP firewall forward rules, default policy: deny
type  prot source      destination  ports
acc/m all 192.168.1.0/24  0.0.0.0/0   n/a
```

Briefly, the output indicates that it will masquerade for all protocols from source 192.168.1.* to anywhere for all source and destination ports.

The **ipfwadm** program is primarily used to set the rules for the firewalling code in the Linux 2.x kernels.

Setting up

Masquerading takes a small amount of effort to get configured. Before kernel 2.0.30 the masquerading code was still considered experimental. As such, many pieces of the code were not included with the full distribution, but only included as patches. If you are running a 2.0.x kernel where x is 29 or lower, see the "Code Maturity Level" sidebar for patching the kernel to include masquerading. My kernel is set up running version 2.0.30 with masquerading

included. The configuration options related to masquerading are shown in [Listing 1](#).

You'll notice that parts of the masquerading code can only be included as kernel modules. These are for specific protocols or applications; examples include FTP, VDOLive and Real Audio.

Currently, IP Masquerading has support for TCP and UDP, FTP, Real Audio, IRC, ICMP (configuration option), VDOLive, CUSeeMe and Quake. Normal TCP and UDP applications like TELNET or DNS are supported directly in the kernel masquerading code. The abnormal (i.e. pathological) protocols such as FTP and IRC (which have IP addresses as a part of the message stream) and Real Audio (because the control protocol needs to know where you are) are supported via kernel modules. VDOLive, CUSeeME and Quake are also kernel modules.

Some Considerations

The `ipfwadm` command, as mentioned, is used to set up the forwarding rules for masquerading. It's also used for creating the firewall and accounting rules also supported by the Linux kernel. While we won't worry about that aspect here, it should be noted that firewall administration and IP masquerading can be tightly coupled depending upon what task you are attempting.

As well as setting the rules, `ipfwadm` allows you to view the current status of any masqueraded connections. For example, the command:

```
# ipfwadm -M -l -n
```

gives the following output:

```
IP masquerading entries
prot expire  source      destination  ports
udp  03:31.50  192.168.1.5  164.109.1.3  1110 (61009) -53
udp  03:36.67  192.168.1.5  164.109.1.3  1112 (61011) -53
tcp  14:06.91  192.168.1.5  207.79.74.21 1304 (61016) -80
tcp  14:07.17  192.168.1.5  207.79.74.21 1303 (61015) -80
tcp  14:05.62  192.168.1.5  207.79.74.21 1302 (61014) -80
tcp  01:44.79  192.168.1.5  204.192.48.81 1301 (61013) -23
```

The `-M` option is used for masquerading administration and is only used with the `-l` or `-s` options. The `-l` option provides the selected (in this case, masquerading) list. As you can see, the output listing shows the originating port number and the masqueraded port number in parentheses going to the destination port. Our traffic shows three DNS queries, three HTTP connections and a TELNET connection currently established.

The expire column pertains to the timers associated with masquerading. A masqueraded packet will have an expiration time if no further traffic is seen. TCP connections have two timers. The first one for the masqueraded

connection and the second one for when the FIN segment is received. Each of these timers can be set with the **-s** option of `ipfwadm`. TCP connections have a default timeout value of 15 minutes and UDP, a timeout value of 5 minutes. You can change the values by giving a command such as:

```
# ipfwadm -M -s 1200 60 120
```

This command signals that TCP connections expire after 20 minutes of inactivity, a 1 minute expiration after receiving the FIN segment and 2 minutes for UDP entries. The default values will, in most cases, work well.

When I added the rule to masquerade the 192.168.1.0 network, I could have indicated which interface masqueraded traffic would be sent and received. I could have extended the command as follows:

```
ipfwadm -F -a m -S 192.168.1.0/24 \  
-D 0.0.0.0/0 -W ppp0
```

ppp0 is the interface name for the 204.192.48.109 address. While this name is optional for our example, it becomes necessary to give it if the masquerading machine has more than one internal network connection. For example, suppose `deathstar` had two Ethernet connections for networks 192.168.1.0 and 192.168.2.0. Without the **-W** option, traffic between the two internal networks would also be masqueraded causing a great deal of confusion.

I chose to deny all packets that were to be forwarded as my default policy using the command:

```
ipfwadm -F -p deny
```

You are allowed to set up the default policy as **masquerade**:

```
# ipfwadm -F -p masquerade
```

This command masquerades all connections going out and those coming in. In other words, if a host on the network external to the masquerading machine sets up a route to the internal network, they can send packets and set up connections (even ping) to internal machines. The danger is clear—your network is no longer hidden.

A certain problem did arise with the 2.0.30 kernel and `ipfwadm`. The IP masquerading code includes entries in the `/proc/net/ip_masquerade` file for ICMP as shown in [Listing 2](#). `ipfwadm` has a problem with this and will give the following error:

```
# ipfwadm -M -l -n  
IP masquerading entries
```

```
ipfwadm: unexpected input data
Try ipfwadm -h for more information.
```

This problem will be taken care of in the near future.

One last option to ipfwadm that I find useful is the **-o** option, which turns on kernel-level debugging for matched packets according to rules set by **ipfwadm**. Note however, this option is only effective when the kernel option for **CONFIG_IP_FIREWALL_VERBOSE** is selected.

Protocols

As we've noted, some protocols are simply incompatible with the basic concept of masquerading. Support exists for a few of the more popular ones—FTP, CUSeeMe, VDOLive, IRC and Quake. A few applications/protocols still exist that are not supported by the 2.0.30 distribution. These include talk, the “R” programs such as **rsh** or **rlogin**, Mplayer and other games.

A great replacement for the “R” programs is **ssh** (Secure Shell). It functions as a direct replacement for rsh, rlogin and **rcp**, as well as support for X11 sessions. Its main provision is for secure communications and strong authentication; however, since it is TCP based, it works quite well with masqueraded hosts. This program is highly recommended for remote communications in general.

IP Masquerading does a very good job of hiding the internal network. If it is hidden, no one can get to you, right? Maybe this isn't a particularly desirable situation. Incoming connections can make sense for resources such as a web server, anonymous FTP server or e-mail. It could be that your Linux host could function for all of these services, and more than likely you would want to proxy incoming mail anyway. If you have a requirement for an internal web server, a couple of programs are available to forward or redirect traffic bound from the external host to an internal host. The simplest is the **redir** program updated by Nigel Metheringham and available from sunsite.unc.edu:/pub/Linux/system/Network/daemons/redir-0.7.tar.gz. **redir** is a TCP port redirector that resides on your masquerading host waiting for connections on a port and redirected to an internal server. A simple example to redirect HTTP and log all connections is:

```
# redir --syslog 192.168.1.5 80 80 &
```

You can also start redir from inetd for more convenience.

Every once in a while I'll have X terminals redirected from machines on the outside sent to “yoda” internally (when I'm not running ssh). I typically have X running on “deathstar”, so I use ports above 6000 for the redirection. For example, I set up the redirection in this way:

```
# redir --syslog 192.168.1.5 6001 6000 &
```

And on the external host I would send the xterm display to port 6001.

```
xterm -display 204.192.48.109:1.0 &
```

You wouldn't want to open this up to everyone, so it's probably a good idea to implement a couple of firewall rules about incoming traffic to the masquerading host to restrict who could connect to you or start redir from inetd and use something like tcpd to restrict connections.

For UDP traffic there are a few programs available to redirect traffic from external to internal hosts. udpspoof (<http://www.america.com/~chrisf/web/udpspoof.c>) and udprelay (<ftp://ftp.wang.com/pub/fitz/udprelay-0.2.tar.Z>) are good ones. Probably the most popular though is ipautofw, which is generic enough to handle both TCP and UDP traffic. It is implemented in the Linux kernel as a part of the 2.0.30 kernel, and it is also a command interface to set up auto-forwarding rules. It's available from <ftp://ftp.netis.com/pub/members/rlynch/ipautofw.tar.gz>.

The ipautofw command sets rules in the /proc/net/ip_autofw file to automatically forward packets for masqueraded hosts. For example, in order to handle Real Audio traffic before the kernel module was implemented, you could use ipautofw to give the command:

```
ipautofw -A -r udp 6970 7170 -c tcp 7070
```

This associated a TCP control connection (-c<!s>7070) with a range of return UDP packets, ports 6970 through 7170 inclusive.

Now, suppose we had that web server behind our masquerading host. We could add an auto-forwarding entry such as

```
# ipautofw -A -r tcp 80 80 -h 195.168.1.5
```

This command forwards HTTP requests to yoda (195.168.1.5). However, ipautofw is only able to fulfill one request at a time. As long as a masquerade entry exists for that connection, all other connection requests are sent TCP "resets" until the timer expires. This is not a good scenario for a web server.

Redirecting traffic to more than one host is another problem with masqueraded redirections. Many sites have multiple web servers internally and use them to load balance the traffic. None of the utilities I've listed so far support this situation.

Fragmentation

IP Fragmentation used to be a problem with masqueraded connections. Because the follow-on fragments contained no transport (TCP or UDP) header information, the datagram could not be correlated to an existing connection. The option for “always defragment” from the configuration menu is shown here:

```
IP: always defragment (CONFIG_IP_ALWAYS_DEFRAG)
[N/y/?] (NEW) y
```

This option will cause fragmented datagrams to be reassembled at the masquerading host rather than at the destination.

Still, the notion of performing reassembly in the “middle” of data transfer goes against the general principle of IP delivery. Other methods exist to help eliminate fragmentation. Two of these are MSS negotiation and path MTU discovery.

MSS (maximum segment size) negotiation is an area where IP Masquerading could improve. What I mean by this is best shown in an example.

From Figure 1, let's look at the TCP traffic generated during a connection open from the machine called falcon to an external machine. Listings 3 and 4 shows the traffic.

Listing 3. TCP three-way hand-shake traffic from falcon to deathstar on 192.168.1.0.

Listing 4

As we can see, the MSS advertised from the masqueraded connection (1326) is exactly the same as the one sent from the original host, falcon. The catch here is how I set up the PPP connection from deathstar to the PPP server. I set the MTU to be 296 knowing fragmentation would occur if the connection was not handled properly. A method of handling that takes advantage of the MSS to eliminate fragmentation is for deathstar, the masquerading host, to readjust the MSS based on knowledge it has of the next hop connection. An MSS of 256 (i.e. 296-40) is more appropriate.

You may also notice the **(DF)** field in the traffic from Figure 4. This is the “Don't Fragment” bit in the IP header. It indicates that if a datagram must be fragmented on its way to the destination, it will be discarded and an ICMP error message sent back to the source. Path MTU discovery is usually responsible for setting the **DF** bit. It does so in order to look for those ICMP error messages, and if found, it will adjust how much data is put into a TCP segment and resend.

A host will continue to do this until a segment size can be found such that the Maximum Transmission Units of all of the data links between the source and destination can accommodate the datagrams without fragmentation.

Given the multitude of methods for overcoming fragmentation, it is no longer a problem in masqueraded networks.

Network Address Translation

A network address translator (NAT) device is one which performs address hiding. A NAT works on relationships that can be 1:1, many:1 or many:n; it also allocates addresses for external use statically or dynamically.

Instead of having a single ISP assigned IP address, a user or company may have an entire class-C, address space. However, the internal network might still be large. With NAT, addresses could be assigned for different functions. For example, assume 199.1.1.0 is the address we have. Further, assume we are using 172.16.0.0 through 172.31.255.255 for our internal networks. We could assign the following:

- 199.1.1.1-10: permanent addresses assigned outside of NAT device
- 199.1.1.11-25: statically assigned to corresponding internal addresses 172.16.1.11-25
- 199.1.1.26-254: dynamically assigned to remaining internal addresses

IP Masquerading is a many:1, static allocation case of NAT

Linux 2.1.x kernels now have support for NAT as a marked entry in the routing table. A full-featured implementation is rumored to be implemented sometime during the 2.1.x development. For many of the abnormal protocols that IP Masquerading already supports, NAT will have to go through the same growing pains. However, it should lead to a more feature-rich and flexible address-hiding environment.

Conclusions

Although the basic functionality has stayed the same, IP Masquerading has progressed tremendously in the past year. Support for new protocols and better handling of old problems have evolved it from experimental status to a fully functional part of Linux kernels. It solves many real world problems for many people and will continue to do so. During the evolutionary development of Linux it may happen that NAT will replace the work that has preceded it, but until then I highly recommend this amazing piece of technology.

Code Maturity Levels

[Glossary](#)

[Resources](#)

Chris Kostick is a Senior Computer Scientist at Computer Sciences Corporation's Network Security Department. He enjoys working with Linux, beginning with kernel version 1.1.18. As far as computers go, he's not sure if he has more fun debugging TCP/IP problems or shooting DOS machines. He can be reached at ckostick@csc.com or by just yelling "Chris" real loud.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Linux Kernel Installation

David Bandel

Issue #43, November 1997

Here's a Kernel Korner geared for the new user. Mr. Bandel guides us through the steps of configuring and compiling the kernel.

Linux is many users' introduction to a truly powerful, configurable operating system. In the past, a Unix-like operating system was out of reach for most. If it wasn't the operating system's 4-digit price tag, it was the hardware. Even the now free-for-personal-use SCO Unixware requires a system with SCSI drives, and most of us are using IDE to keep costs down. Along with the power that Linux brings comes the need to perform a task users have not had to do on simpler operating systems: configure the kernel to your hardware and operations.

Previous installation kernels from 1.2.x and before *suggested* that you rebuild; however, with the new 2.0.x kernel, rebuilding has almost become a necessity. The kernel that comes with the installation packages from Red Hat, Caldera, Debian and most others, is a generic, "almost everything is included" kernel. While rebuilding a kernel may seem like a daunting task and living with the installed kernel may not be too bad, rebuilding is a good introduction to your system.

Why Roll Your Own?

The standard installation kernels are an attempt to make as many systems as possible usable for the task of installing a workable Linux system. As such, the kernel is bloated and has a lot of unnecessary code in it for the average machine. It also does not have some code a lot of users want.

Then, of course, there's always the need to upgrade the kernel because you've bought new hardware, etc. Upgrading within a series is usually very straightforward. When it comes to upgrading, say from 1.2.something to 2.0.something, now the task is beyond the scope of this article and requires

some savvy. Better to get a new distribution CD and start fresh—this is also true for upgrading to the experimental 2.1.x kernels.

Kernel Version Numbering

All Linux kernel version numbers contain three numbers separated by periods (dots). The first number is the kernel version. We are now on the third kernel version, 2. Some of you may be running a version 1 kernel, and I am aware of at least one running version 0 kernel.

The second number is the kernel major number. Major numbers which are even numbers (0 is considered an even number) are said to be stable. That is, these kernels should not have any crippling bugs, as they have been fairly heavily tested. While some contain small bugs, they can usually be upgraded for hardware compatibility or to armor the kernel against system crackers. For example, kernel 2.0.30, shunned by some in favor of 2.0.29 because of reported bugs, contains several patches including one to protect against SYN denial of service attacks. The kernels with odd major numbers are developmental kernels. These have not been tested and often as not will break any software packages you may be running. Occasionally, one works well enough that it will be adopted by users needing the latest and greatest support before the next stable release. This is the exception rather than the rule, and it requires substantial changes to a system.

The last number is the minor number and is increased by one for each release. If you see kernel version 2.0.8, you know it's a kernel 2.0, stable kernel, and it is the ninth release (we begin counting with 0).

Assumptions

I hate to make any assumptions; they always seem to come back to bite me. So I need to mention a few things so that we're working off the same sheet of music. In order to compile a kernel, you'll need a few things. First, I'll assume you've installed a distribution with a 2.0.x kernel, all the base packages and perhaps a few more. You'll also need to have installed **gcc** version 2.7 and all the supporting gcc libraries. You'll also need the **libc-dev** library and the **binutils** and **bin86** packages (normally installed as part of a standard distribution install). If you download the source or copy it from a CD, you'll also need the **tar** and **gunzip** packages. Also, you'll need lots of disk real estate. Plan on 7MB to download, another 20MB to unpack this monster and a few more to compile it.

Needless to say, many of the things we will discuss require you to be logged in as root. If you've downloaded the kernel as a non-privileged user and you have write permission to the `/usr/src` subdirectory, you can still do much of this task without becoming root. For the newcomers to Linux, I highly recommend you

do as much as possible as a non-privileged user and become root (type: **su - face**) only for those jobs that require it. One day, you'll be glad you acquired this habit. Remember, there are two kinds of systems administrators, those who have totally wrecked a running setup inadvertently while logged in as root, and those who will.

Obtaining/Upgrading the Source

Kernel sources for Linux are available from a large number of ftp sites and on almost every Linux distribution CD-ROM. For starters, you can go to ftp.funet.fi, the primary site for the Linux kernel. This site has a list of mirror sites from which you can download the kernel. Choosing the site nearest you helps decrease overall Internet traffic.

Once you've obtained the source, put it in the /usr/src directory. Create a subdirectory to hold the source files once they are unpacked using tar. I recommend naming the directories something like linux-2.0.30 or kernel-2.0.30, substituting your version numbers. Create a link to this subdirectory called linux using the following command:

```
ln -sf linux-2.0.30 linux
```

I included the **-f** in the link command because if you already have a kernel source in /usr/src, it will contain this link too, and we want to force it to look in our subdirectory.¹ The only time you may have a problem is if linux is a subdirectory name, not a link. If you have this problem, you'll have to rename the subdirectory before continuing:

```
mv linux linux-2.0.8
```

Now issue the command:

```
tar xzvf linux-kernel-source.tar.gz
```

I have a habit of always including **w** (wait for confirmation) in the tar option string, then when I see that the .tar.gz or .tgz file is going to unpack into its own subdirectory, I **ctrl-C** out and reissue the command without the **w**. This way I can prevent corrupted archives from unpacking into the current directory.

On some versions of **ln** (notably version 3.13), the force option (**-f**) does not work. You'll have to first remove the link then establish it again. This works correctly by version 3.16.

Once you have the kernel unpacked, if you have any patches you wish to apply, now is a good time. Let's say you don't wish to run kernel 2.0.30, but you do want the tcp-syn-cookies. Copy the patch (called tcp-syn-cookies-patch-1) into the /usr/src directory and issue the command:

```
patch < tcp-syn-cookies-patch-1
```

This command applies the patch to the kernel. Look for files with an .rej extension in the /usr/src directory. These files didn't patch properly. They may be unimportant, but peruse them anyway. If you installed a Red Hat system with some but not all of the kernel source (SPARC, PowerPC, etc.), you'll see some of these files. As long as they're not for your architecture, you're okay.

Preparation

As a final note, before we change (**cd**) into the kernel source directory and start building our new kernel, let's check some links that are needed. In your /usr/include directory, make sure you have the following soft links:

```
asm - /usr/src/linux/include/asm
linux - /usr/src/linux/include/linux
scsi - /usr/src/linux/include/scsi
```

Now, you see another reason to standardize the location of the kernel. If you don't put the latest kernel you wish to install in /usr/src/linux (via a link), the above links will not reach their intended target (dangling links), and the kernel may fail to compile.

How to proceed

Once everything else is set up, change directories into /usr/src/linux. Although you may want to stop off and peruse some of the documentation in the Documentation directory, particularly if you have any special hardware needs. Also, several of the CD-ROM drivers need to be built with customized settings. While they usually work as is, these drivers may give warning messages when loaded. If this doesn't bother you and they work as they should, don't worry. Otherwise, read the appropriate .txt, .h (header) files and .c (c code) files. For the most part, I have found them to be well commented and easy to configure. If you don't feel brave, you don't have to do it. Just remember you can always restore the original file by unpacking the gzipped tar file (or reinstalling the .rpm files) again.

Beginning to Compile

The first command I recommend you issue is:

```
make mrproper
```

While this command is not necessary when the kernel source is in pristine condition, it is a good habit to cultivate. This command ensures that old object files are not littering the source tree and are not used or in the way.

Configuring the Kernel

Now, you're ready to configure the kernel. Before starting, you'll need to understand a little about modules. Think of a module as something you can plug into the kernel for a special purpose. If you have a small network at home and sometimes want to use it (but not always), maybe you'll want to compile your Ethernet card as a module. To use the module, the machine must be running and have access to the `/lib/modules`. This means that the drive (IDE, SCSI, etc., but could be an ethernet card in the case of `nfs`), the file system (normally `ext2` but could be `nfs`) and the kernel type (hopefully `elf`) must be compiled in and cannot be modules. Modules aren't available until the kernel is loaded, the drive (or network) accessed, and the file system mounted. These files must be compiled into the kernel or it will not be able to mount the root partition. If you're mounting the root partition over the network, you'll need the network file system module, and your Ethernet card compiled.

Why use modules? Modules make the kernel smaller. This reduces the amount of protected space never given up by the kernel. Modules load and unload and that memory can be reallocated. If you use a module more than about 90% of the time the machine is up, compile it. Using a module in this case can be wasteful of memory, because while the module takes up the same amount of memory as if it were compiled, the kernel needs a little more code to have a hook for the module. Remember, the kernel runs in protected space, but the modules don't. That said, I don't often follow my own advice. I compile in: `ext2`, IDE and `elf` support only. While I use an Ethernet card almost all the time, I compile everything else as modules: `a.out`, `java`, `floppy`, `iso9660`, `msdos`, `minix`, `vfat`, `smb`, `nfs`, `smc-ultra` (Ethernet card), `serial`, `printer`, `sound`, `ppp`, etc. Many of these only run for a few minutes at a time here and there.

The next step is to configure the kernel. Here we have three choices—while all do the same thing, I recommend using one of the graphical methods. The old way was to simply type: **make config**. This begins a long series of questions. However, if you make a mistake, your only option is to press **ctrl-C** and begin again. You also can't go back in the sequence, and some questions depend on previous answers. If for some reason you absolutely can't use either of the graphical methods, be my guest.

I recommend using either **make menuconfig** or **make xconfig**. In order to use **menuconfig**, you must have installed the `ncurses-dev` and the `tk4-dev` libraries. If you didn't install them and you don't want to use the next method, I highly recommend that you install them now. You can always uninstall them later.

To run **make xconfig**, you must install and configure X. Since X is such a memory hog, I install, configure and **startx** only for this portion of the process, going back to a console while the kernel compiles so it can have all the memory

it needs. The xconfig menu is, in my opinion, the best and easiest way to configure the kernel. Under menuconfig, if you disable an option, any subordinate options are not shown. Under xconfig, if you disable an option, subordinate options still show, they are just greyed out. I like this because I can see what's been added since the last kernel. I may want to enable an option to get one of the new sub-options in order to to experiment with it.

I'm going to take some space here to describe the sections in the kernel configuration and tell you some of the things I've discovered—mostly the hard way.

The first section is the code-maturity-level option. The only question is whether you want to use developmental drivers and code. You may not have a choice if you have some bleeding edge hardware. If you choose “no”, the experimental code is greyed out or not shown. If you use this kernel for commercial production purposes, you'll probably want to choose “no”.

The second section concerns modules. If you want modules, choose “yes” for questions 1 and 3. If you want to use proprietary modules that come with certain distributions, such as Caldera's OpenLinux for their Netware support, also answer “yes” to the second question since you won't be able to recompile the module.

The third section is general setup. Do compile the kernel as ELF and compile support for ELF binaries. Not compiling the proper support is a definite “gotcha”. You'll get more efficient code compiling the kernel for the machine's specific architecture (Pentium or 486), but a 386 kernel will run in any 32-bit Intel compatible clone; a Pentium kernel won't. An emergency boot disk for a large number of computers (as well as distribution install disks) is best compiled as a 386. However, a 386 will not run a kernel compiled for a Pentium.

Next comes block devices—nothing special here. If your root device is on an IDE drive, just make sure you compile it.

Then comes networking. For computers not connected to a network, you won't need much here unless you plan to use one computer to dial-out while others connect through it. In this case, you'll need to read up on such things as masquerading and follow the suggested guidelines.

SCSI support is next, though why it doesn't directly follow block devices I don't know. If your root partition is on a SCSI device, don't choose modules for SCSI support.

SCSI low-level drivers follow general SCSI support. Again, modules only for devices that don't contain the root partition.

The next section takes us back to networking again. Expect to do a lot of looking for your particular card here as well as some other support such as **ppp**, **slip**, etc. If you use **nfs** to mount your root device, compile in Ethernet support.

For those lucky enough to be needing ISDN support, the ISDN subsection will need to be completed.

Older CD-ROMs may require support from the next section. If you're using a SCSI or IDE CD-ROM, you can skip this one.

Next comes file systems. Again, compile what you need, in most cases ext2 and use modules for the rest.

Character devices are chosen next. Non-serial mice, like the PS/2 mouse are supported. Look on the bottom of your mouse. Many two-button mice are PS/2 type, even though they look and connect like serial mice. You'll almost certainly want serial support (generic) as a minimum. Generic printer support is also listed here.

The penultimate section is often the most troubling: sound. Choose carefully from the list and read the available help. Make sure you've chosen the correct I/O base and IRQs for your card. The MPU I/O base for a SoundBlaster card is listed as 0. This is normally 330 and your sound module will complain if this value is incorrect. Don't worry. One of the nice things about modules is you can recompile and reinstall the modules as long as the kernel was compiled with the hook. (Aren't modules great?).

The final section contains one question that should probably be answered as "no, kernel hacking".

Save your configuration and exit.

I have, on several occasions, had trouble editing the numbers in menuconfig or xconfig to values I knew were correct. For whatever reason, I couldn't change the number or **config** wouldn't accept the number, telling me it was invalid. For example, changing the SoundBlaster IRQ from the config default of 7 to 5, and the MPU base I/O from 0 to 300. If you experience this problem, but everything else went well, don't despair. The file you just wrote when you did a "Save" and "Exit" is an editable text file. You may use your text editor of choice: Emacs, vi, CrispLite, joe, etc. Your configuration file is in the /usr/src/linux directory and is called **.config**. The leading dot causes the file to be hidden during a normal

directory listing (**ls**), but it shows up when the **-a** option is specified. Just edit the numbers in this file that you had trouble with in the configuration process. Next, type **make dep** to propagate your configurations from the `.config` file to the proper subdirectories and to complete the setup. Finally, type **make clean** to prepare for the final kernel build.

Building the Kernel

We're now ready to begin building the kernel. There are several options for accomplishing this task:

- **make zImage**: makes the basic, compressed kernel and leaves it in the `/usr/src/linux/arch/i386/boot` directory as `zImage`.
- **make zlilo**: Copies the `zImage` to the root directory (unless you edited the top-level Makefile) and runs LILO. If you choose to use this option, you'll have to ensure that `/etc/lilo.conf` is preconfigured.
- **make zdisk**: Writes `zImage` to a floppy disk in `/dev/fd0` (the first floppy drive—the `a:` drive in DOS). You'll need the disk in the drive before you start. You can accomplish the same thing by running **make zImage** and copying the image to a floppy disk `cp /usr/src/linux/arch/i386/boot/zImage /dev/fd0`. Note that you'll need to use a high-density disk. The low density 720k disks will reportedly not boot the kernel.
- **make boot**: Works just the same as the `zImage` option.
- **make bzImage**: Used for big kernels and operates the same as `zImage`. You will know if you need this option, because **make** will fail with a message that the image is too big.
- **make bzdisk**: Used for big kernels and operates the same as `zdisk`. You will know if you need this option, because **make** will fail with a message that the image is too big.

Other **make** options are available, but are specialized, and are not covered here. Also, if you need specialized support, such as for a RAM disk or SMP, read the appropriate documentation and edit the Makefile in `/usr/src/linux` (also called the top-level Makefile) accordingly. Since all the options I discussed above are basically the same as the `zImage` option, the rest of this article deals with **make zImage**--it is the easiest way to build the kernel.

For those of you who wish to speed up the process and won't be doing other things (such as configuring other applications), I suggest you look at the man page for **make** and try out the **-j** option (perhaps with a limit like 5) and also the **-l** option.

If you chose modules during the configuration process, you'll want to issue the commands:

```
make modules
make modules_install
```

to put the modules in their default location of `/lib/modules/2.0.x/`, `x` being the kernel minor number. If you already have this subdirectory and it has subdirectories such as `block`, `net`, `scsi`, `cdrom`, etc., you may want to remove `2.0.x` and everything below it unless you have some proprietary modules installed, in which case don't remove it. When the modules are installed, the subdirectories are created and populated.

You could just as easily have combined the last three commands:

```
make zImage; make modules; make modules_install
```

then returned after all the disk churning finished. The `;` (semicolon) character separates sequential commands on one line and performs each command in order so that you don't have to wait around just to issue the next command.

Once your kernel is built and your modules installed, we have a few more items to take care of. First, copy your kernel to the root (or `/boot/` or `/etc/`, if you wish):

```
cp /usr/src/linux/arch/i386/boot/zImage /zImage
```

You should also copy the `/usr/src/linux/System.map` file to the same directory as the kernel image. Then change (`cd`) to the `/etc` directory to configure LILO. This is a very important step. If we don't install a pointer to the new kernel, it won't boot. Normally, an install kernel is called `vmlinuz`. Old-time Unix users will recognize the construction of this name. The trailing "z" means the image is compressed. The "v" and "m" also have significance and mean "virtual" and "sticky" respectively and pertain to memory and disk management. I suggest you leave the `vmlinuz` kernel in place, since you know it works.

Edit the `/etc/lilo.conf` file to add your new kernel. Use the lines from the **`image=vmlinuz`** line to the next **`image=`** line or the end. Duplicate what you see, then change the first line to **`image=/zImage`** (assuming your kernel is in the root directory) and choose a different name for the **`label=`**. The first image in the file is the default, others will have to be specified on the command line in order to boot them. Save the file and type:

```
lilo
```

You will now see the kernel labels, and the first one will have an asterisk. If you don't see the label that you gave your new kernel or LILO terminates with an error, you'll need to redo your work in `/etc/lilo.conf` (see LILO man pages).

We're almost ready to reboot. At this point, if you know your system will only require one reboot to run properly, you might want to issue the command:

```
depmod -a 2.0.x
```

where *x* is the minor number of the kernel you just built. This command creates the dependencies file some modules need. You'll also want to make sure you don't boot directly into **xdm**. For Red Hat type systems, this means ensuring the `/etc/inittab` file doesn't have a default run level of 5, or that you remember to pass LILO the run level at boot time. For Debian systems, you can just type:

```
mv /etc/init.d/xdm /etc/init.d/xdm.orig
```

for now and move it back later.

Normal Rebooting the New Kernel

Reboot your machine using:

```
shutdown -r now
```

While typing **reboot** or pressing the **ctrl+alt+del** key combination usually works, I don't recommend either one. Under some circumstances, the file systems won't be properly unmounted and could corrupt open files. At the LILO prompt, if you need to boot the old kernel or pass some parameters for bootup and you don't see the **boot:** prompt, you can try pressing either the **shift** or **ctrl** key, and the **boot:** prompt should appear. Once you have it, press **tab** to see the available kernel labels. Type the label and optionally enter any parameters for bootup. Normally, however, the default kernel should boot automatically after the timeout interval specified in the `/etc/lilo.conf` file. During bootup, you may see a few error messages containing: `SIOCADDR` or the like. These usually indicate that a module (normally a network module) didn't load. We'll handle this shortly. If you got the error, "VFS, cannot mount root", you didn't compile the proper disk or file-system support into the kernel.

Troubleshooting

Due to the different ways in which each distribution handles daemon startup from `/etc/inittab`, it is difficult in this article to cover all the possible reasons your bootup may not have gone smoothly and the reasons why. However, I can tell you where to start looking.

First, run **depmod -a** to ensure you have an up-to-date, module dependency file (it will be created in the appropriate subdirectory). If you get a string of errors about unresolved dependencies, old modules are present in the modules subdirectories, and you didn't configure the kernel with "Module Versions"

enabled. This is not a fatal error. The modules you compiled and installed are good. Check the `/etc/conf.modules` file and make sure that any lines pointing to `/lib/modules` are complete:

```
/lib/modules/`uname -r`/xx
```

(Note: the grave quote on each side of **uname -r** is located above the Tab key in the upper left corner of the keyboard on a U.S. keyboard).

Make sure **kerneld** is running and that it is loaded early in the bootup process. If it is, then the system doesn't need to explicitly load modules, kerneld will handle it. Be careful about calling kerneld too early in the first rc script. kerneld will stop the bootup process forcing a hard reboot via the reset button or power switch, if it is called before the system knows its host name. If this happens to you, you can reboot passing LILO the **-b** argument which prevents **init** from executing any rc scripts. Next, look in `/etc/rc.d/` at the `rc`, `rc.sysinit` and `rc.modules` files. One or more may point to a directory such as `/etc/modules/`uname -r`/`uname -v`` where a list of bootup modules are located. You can just copy the old file over to the new directory;

```
mkdir /etc/modules/`uname -r` ;  
cp /etc/modules/2.0.xx/g#1 Thu 3 Sep 1997.\  
default /etc/modules/`uname -r`/\  
`uname -v`.default"
```

Your system will almost certainly have a different date for the modules file. Your system also may or may not use the default extension. Pay close attention to the use of grave quotes and double quotes in the above example, since both are needed in the proper places. Once you have found the keys to your system, you should be able to reboot into a properly functioning system. If you experience further problems, the best place to get quick, expert advice is on a mailing list dedicated to your particular distribution. Those successfully running a particular distribution usually delight in assisting novices with problems they may encounter. Why? Because they hit the same brick walls when they were novices and received help with many problems. Lurk a few days on a list, and if your question isn't asked by someone else, ask it yourself. Check the mail-list archives first, if any are present. These archives contain answers to frequently asked questions (FAQ).

Conclusion

While building a kernel tailored to your system may seem a daunting challenge for new administrators, the time spent is worth it. Your system will run more efficiently, and more importantly, you will have the satisfaction of building it yourself.

The few areas where you may encounter trouble are in remembering to rerun LILO after installing the new kernel, but you didn't overwrite your old one (or did you?), so you can always revert to one that worked from the **lilo:** prompt. Distribution specific problems during bootup may also be encountered during the first reboot but are usually easily resolved. Help is normally only an e-mail away for those distributions that don't come with technical support.

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

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Using SAMBA to Mount Windows 95

Jonathon Stroud

Issue #43, November 1997

This article presents a script for easy and efficient mounting of shared directories using SAMBA.

Many major universities are now offering network connections to students in their rooms. This is a wonderful opportunity for the Linux community. Even though the majority of student-owned computers on these networks are still running Windows 95, many students are making the switch to Linux. One question newcomers to Linux are constantly asking is, "Can I access a directory shared by a Windows 95 computer in the *Network Neighborhood*, and can I share files to Windows 95 users?" The answer, of course, is "Yes." I keep telling them there is nothing that Linux cannot do, yet they continue to come to me and ask if they can do this or that in Linux. I have never once answered "No".

SAMBA

To mount a Windows 95 share, we use a program called SAMBA. SAMBA is a program that allows Linux to talk to computers running Windows for Workgroups, Windows 95, Windows NT, Mac OS and Novel Netware. SAMBA even allows you to share a printer between computers using these different operating systems. SAMBA comes with most distributions of Linux, but if you do not have it installed, you can download it from the SAMBA home page at <http://lake.canberra.edu.au/pub/samba/>.

Mounting Windows 95 Shares

The first thing to do is check to see which directories are shared on the computer you are using. To do this, type:

```
smbclient -L
```

This command lists all of the shared directories. To mount the desired directory, use the command **smbmount**, which can be a bit tricky. I have

created a script, named **smb**, that allows users to mount drives using `smbmount` with relative ease. That script is shown in [Listing 1. smb Script](#)

To execute this script you simply type **smb** followed by the name of your computer and the directory you wish to mount (for example, **smb workstation files**). If you are root, the script creates a directory in `/mnt` with the same name as the computer and mounts the directory in that location. For any other user, the script creates a directory in the users home directory named `/mnt`. In that directory, `smb` creates another directory with the same name as the computer and mounts the shared directory there.

Sharing files with Windows 95

Sharing a file is not too difficult. To share a directory you need to edit the `/etc/smb.conf` file. By default, Samba shares users' home directories, but they are only visible (and accessible) to the owner. This means the person accessing the shared files must be logged into Windows 95 with the same login ID as he used to log into the Linux box.

In order to let the user bob and only the user bob access the directory `/shares/files`, add the following lines to your `/etc/smb.conf` file:

```
1 [bobsfiles]
2 comment = files for bob
3 path = /shares/files
4 valid users = bob
5 public = no
6 writable = yes
7 printable = no
```

Here's a line by line look at this example.

1. Specifies the name to be used for the shared directories.
2. Specifies a comment to be displayed in the Windows 95 Network Neighborhood.
3. Specifies the name of the directory on your computer to be shared.
4. Sets bob as the only valid user.
5. Specifies no public access. When set to **yes**, users are allowed to access the directory with guest privileges.
6. Indicates whether or not the user has write permissions to the shared directory.
7. Specifies that file cannot be printed. When set to **yes**, users are allowed to spool print jobs from the shared directory.

More examples on sharing files can be found in the default `smb.conf` file. For more help on setting up this file, see the Samba web page or type:

```
man smb.conf
```

Another cool Samba Option

If a Windows 95 user on your network is running **winpopup** (an instant messaging program), you can send them a winpopup message using Samba. To do this just type:

```
smbclient -M
```

and the contents of *message_text* will be displayed in a message window on *computername*.

This article was first published in Issue 19 of LinuxGazette.com, an on-line e-zine formerly published by Linux Journal.



Jonathan Stroud (jgstroud@eos.ncsu.edu) is a full-time student at North Carolina State University. He enjoys helping users switch their primary operating system from Windows to Linux and proving that you no longer have to be a Computer Scientist to use Linux.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Best of Technical Support

Various

Issue #43, November 1997

Our experts answer your technical questions.

Forgotten Password

I cannot log on. My old password doesn't work. I am the root on this system. How can I change the password or verify my old password? —Kevin Cary Red Hat 2.0.30

This is a much debated topic since it essentially involves breaking your system's security. The main discussion point always seems to be how to do it without letting people do it to others. For the most part, it is easy to remove the root password from Linux (and many other flavors of Unix) servers, as long as the machine has a floppy drive that it boots from by default.

If you have a standard Linux system without shadow passwords, you can simply boot from a "boot and root" floppy disk set, mount your partition and edit the `/etc/passwd` file. In this file, fields are separated by the colon (:) character. The second field on the first line can be removed (so the entry looks like `::`) and root will no longer have a password. You can then reboot, log in as root and use `passwd` to set the new root password. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Choosing A Distribution

Which distribution should I download? Sorry about this low-level question, but I couldn't find help elsewhere. —Erik Rask

That's a question that can easily spark religious debates, but I'll side step that issue and just mention that there's a rather well-written Distribution HOWTO available from any Linux Documentation Project mirror (i.e., <http://www.silug.org/LDP/>) that describes the differences between distributions. —Steven Pritchard steve@silug.org

Slow Modem Connection

I need help in determining why my modem connection is so slow. I have a 33.6Kb modem that runs fine in Windows, but I just haven't found all of the files I must edit for making **pppd** run as fast as possible. —Paul Carff

pppd has a setting called **asynctest**, that is a mask of characters which it must escape. This masking allows **pppd** to be run across lines that may not handle all 256 characters of the ASCII set. A common example of this is a line that must use software flow control in the form of the **ctrl-S** and **ctrl-Q** characters to pace the flow of data. **pppd** cannot send those characters directly since they would interfere with the operation of the flow control; so, it sends another two characters (the escape and the replacement character) instead.

*If no **asynctest** parameter is set (as described in the **pppd** man page), **pppd** will automatically escape all control characters just to be sure the transmission is not interrupted. Obviously, this adds a good deal of overhead to the transmissions. Setting the **asynctest** to 0 results in a noticeable increase in transfer rates on most systems. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com*

Reconfiguring the Mouse

I just installed Red Hat 4.1, and I made a mistake while setting up the mouse. I set the mouse port to 1, but the real mouse port is 2. How can I change the mouse port?

—Moon Ill June

The best way is to run **mouseconfig** from the command line. It will let you choose the port just as you did at install time. —Donnie Barnes, Red Hat Software redhat@redhat.com

Kernel Stack Corruption

I am running 2.0.7 and have tried to compile the kernels of 2.0.27, 2.0.30 and 2.1.x, all of which give me the same error on execution. All of the kernels compiled successfully, but they crashed on bootup with the message "**kernel stack corruption**". What could be wrong? —Thomas S. Chin

Start by using a good memory checker to check your system RAM. The kernels you listed are known to be stable, and the kernel stack is somewhat difficult to corrupt since it's well protected by the operating system.

Also, investigate your BIOS settings to make sure they match your memory type and CPU-cache type. If it can be set, be sure your BIOS has the same speed setting (60ns or 70ns) as your system RAM and your cache type (write-back, write-through, et cetera) matches what you actually have. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Jumbled Text and Setting Color

I like color as a means of segregating data and reducing eye strain. I looked at all the escape sequence information I could find and set PS1 (where _ is a space) to:

```
\033[36m\u_\033[33m\w_\$_ --\033[32m_
```

The user name is cyan, the directory and root prompt are brown and the rest is green. However, when I try to edit a history command, strange things occur when backspacing and the command text becomes jumbled. What am I missing? —Jim Red Hat 4.1

*The shell probably doesn't care what escape sequences you use. The problem area is more likely your terminal. Investigate the terminal program you are using to log in and be sure it supports proper ANSI sequences. Be sure that it properly handles setting a color when another is already set. Never turn off colors at the end of your prompt. Try an **esc[Om** at the end of the line (sacrificing color for the text you type) and see if that helps.*

This is especially true since a colorized **ls** listing causes trouble for you. The listing is turning on and off individual colors (with the default **ls** settings), and terminals that don't support it will have problems with pre-existing color settings. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Setting Time

Although my DEC Alpha NFS server and the Linux machines all have the same local time, every file written to the NFS partition in the Linux machines is one hour (exactly one hour) ahead in time. For example, if both the server and a Linux client have 20:30, and the client writes a file to the NFS partition, the time of creation of the file is 21:30. How can I correct this problem? —Jose Luis Richardo ChavezRed Hat Linux 4.2 and 4.1

Linux adjusts the time displayed by the **date** command for your local time zone. It's possible to see the "correct" time from **date**, if your system's internal notion of the correct time and the time zone setting are both wrong. For example, if your internal clock is an hour fast, but your time zone is an hour behind, you'll

see the behavior you described. Check the time zone output in the date command. For example:

```
Mon Aug 4 12:12:53 PDT 1997
```

indicates we are currently running Pacific Daylight Time. On Red Hat systems you want /etc/localtime to be a link to a file in /usr/lib/zoneinfo.

Also, you generally want your system to store time in your local time. Edit the /etc/sysconfig/clock file, so that you have:

```
UTC=false  
ARC=true
```

Then go into your system's BIOS setup to check that the local time is set correctly. —Larry M. Augustin, VA Research lma@varesearch.com

Running XDM

How do I properly run XDM at system bootup? —Aris Seisums Slackware 1.3.20

I'll assume you've got X and XDM set up properly; if not, read the man page. Most distributions provide an XDM setup that works, so you may be able to just use it. Also, make sure you can use **startx** to get into X after logging on in the normal way.

Next, look for a line like:

```
id:5:initdefault
```

in your /etc/inittab file. This will tell you what run level init starts at. The run level is the number in the middle. Add a line like this:

```
x:5:respawn:/usr/bin/X11/xdm -nodaemon
```

to the /etc/inittab file. For the 5 shown above, substitute the run level you're running at—then reboot.

If you want to test this first (good idea), pick a different run level (one below) to run XDM at. Don't pick 0, 1 or 6 as these have special meanings. Then, as root, run **telinit<\!s>runlevel** to switch to that run level. If it doesn't work, **telinit<\!s>regular<\!s>runlevel** will switch you back. —Jeff Licquia
jeff@web.landscape.net

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.