Linux Journal Issue #52/August 1998

Features

Getting Help With Linux *by Michael Stutz*
: So you've heard a lot about the power of Linux and you're eager to try it out for yourself. But where do you start?

Migrating to Linux, Part 1 *by Norman M. Jacobowitz*
: Linux—not just for hackers anymore...

Betting on Darwin *by Doc Searls*
: Doc Searls interviews Marc Andreessen and Tom Paquin on Netscape's Open Source Strategy.

Selecting a Linux Distribution *by Phil Hughes*

Linux Distributions Table
: Having trouble deciding which distribution to go for? Here's help.

News & Articles

Encapsulating IP Using SCSI *by Ben Elliston*
: Mr. Elliston is working on a protocol for using SCSI devices to network Linux clusters in order to transfer data at high speeds.

A First Look at KDE Programming *by David Sweet*
: Mr. Sweet teaches us how to write an application for the KDE desktop—for the experienced GUI programmer.

Linux Stampede *by David Haraburda*
: This article tells us a bit about the new kid on the block—Stampede Linux.

Muscle Flexes Smart Cards into Linux *by David Corcoran*

System Administration Linux as a Backup E-mail Server *by John Blair*

Linux as a Backup E-mail Server Implementing a fall-back e-mail server is easy to do by setting the proper entries in the DNS server and running sendmail on a Debian Linux system.

Best of Technical Support

Archive Index

Advanced search

Getting Help With Linux

Michael Stutz

Issue #52, August 1998

So you've heard a lot about the power of Linux and you're eager to try it out for yourself. But where do you start?

Whether you're a long-time Linux user or a total newbie who needs to obtain Linux software for that first install, it is easy to become overwhelmed by the vast amount of Linux information available. Finding the best approach to getting help on a particular problem is no simple task. Since Linux is an independent happening—no company or sole entity *owns* what is essentially a continually-growing resource, free for all humanity—this tangle of information can be confusing. Here's an attempt to outline the most efficient means of getting help with Linux.

First, understand that no matter what your skill level, you're not alone. Help is always available—in fact, one of the strengths of the free software movement is that you don't have to wait on a tech support line, or rely on any one business (and local business hours) for help—individuals and companies all over the world can provide all levels of support.

There are four basic routes to getting help with Linux; the one you choose for any particular problem will depend on what that problem is. These routes almost always overlap, and eventually you'll probably have dealt with all four in varying degrees. They are:

- Books and Media
- The Internet
- Regional user groups
- Consultants

Books and Media

If you already have a computer you want to run Linux on, and you're ready to begin fiddling with the hardware settings, then this is a good place to begin. Books are also the best desktop reference to keep handy when you need quick information—and you can take them along when you're away from a computer. For beginners, a good book is quite a deal because it often comes bundled with a Linux CD-ROM.

Note, however, that like any other popular subject, the mileage of any given Linux book will vary—they range from excellent tomes of which a shopworn copy is an absolute *must* for every Linux guru's lair, to those that contain inaccuracies and typos. This same warning also applies to the many Linux CD-ROMs available.

When you are in the market for a CD-ROM, make sure that you are getting a recent distribution; these can be obtained from many vendors, such as Cheap Bytes, Prime Time Freeware and Linux System Labs, often for as little as a few dollars. Be wary of older CD-ROMs; using older versions of the software may be more trouble than they're worth—the active and continual development of the Linux system means that the software on old CD-ROMs will be significantly different from the bleeding edge Linux that's out in the field. Since Linux is constantly getting easier to install and use and hardware compatibilities are always being added, you will save yourself some grief by installing from the newest version you can get.

On that note, be careful of the publication date on any books you might be considering. As a rule, any technical book or CD is outdated to some degree upon publication. It is generally safe to assume that *any Linux book or CD over a year and a half old is hopelessly out of date*. Furthermore, if you find a Linux book or CD more than *three* years old, consider it an object of Internet and computing antiquity—maybe keep it for historical purposes or as a conversation piece.

When it comes to applied computing—and specifically the Internet/UNIX world—the books published by O'Reilly & Associates seem to be without peer. Their *TCP/IP Network Administration* by Craig Hunt is a must-have for learning the basics of Internet networking, and they've done the same with their selection of fine books on Linux. The latest edition of their book *Running Linux*, by Matt Welsh and Lar Kaufman, is perhaps the finest all-around general-purpose Linux overview currently available.

The Free Software Foundation also publishes many books documenting free software, including the GNU Emacs editor; unlike most other publishers of

technical manuals, the FSF's books are as free as the software they write about, and buying these books is a good way to show support for this organization.

As far as CD-ROMs are concerned, you will probably want to stick with those containing either (or both) the Debian or Red Hat distributions of Linux. A *distribution* is necessary; it is simply a collection of the various programs and applications which make up a running, usable Linux system, along with some way of upgrading or maintaining the system. They all contain the same Linux programs and software, but each one collects them differently and has a different means for installation and upgrades.

Slackware was an early distribution that was excellent for its time, but seems to have taken a back seat recently as far as popularity goes. There are many other less popular distributions, each with its own strengths and weaknesses—but for a beginner, sticking with one of the top two (Debian and Red Hat) is probably your best bet. As was pointed out on IRC recently, distributions are to Linux like flavors are to ice cream—there's much more than just vanilla and chocolate out there, and which one you eventually settle down with will depend on your own taste.

Debian GNU/Linux is a free distribution that, like Linux itself, is assembled by a loose collection of enthusiasts. Red Hat is a commercial product (*also* assembled by Linux enthusiasts) that most consider to be much easier to install and maintain.

The Internet

On-line is where all the action is, for sure! Like most free software, Linux was developed in its entirety by a virtual community on the Internet, and the Internet is where you'll always find the latest and greatest in Linux developments. You're missing out on a *lot* if you're not connected to the Internet. The ISP Hookup HOWTO gives all the details.

If you want, you can download Linux right off the Net and onto your computer. I've done this myself, and it's still my preferred method of installation on systems that don't have a CD-ROM drive. But be warned: this could take quite some time with a regular modem connection.

The general path to take when downloading Linux from the Net is first to download a set of installation disks—typically 5-10 diskettes—from which you install a bare-bones Linux system on your computer. Then use your existing Internet connection (a modem and an ISP dial-up account, for instance) to reconnect and download the specific applications and packages that you need. Instructions on how to do this are at both the Red Hat and Debian web sites.

I also like to frequent the 2GB+ Linux software repository at Sunsite, a machine hosted by the University of North Carolina. (There are mirrors of it worldwide.) They also host an excellent hands-on tutorial for total newbies who want to learn how to use Linux.

Not only is the Net where all the developments are taking place, but this is also where some of the best help and documentation can be found. If you want to learn how to do something with Linux, or have a question about some aspect of your Linux system, here is the way to go about finding an answer on the Net (see Resources for addresses):

1. **Check the HOWTOs**. The Linux HOWTOs are part of the Linux Documentation Project and a useful collection of up-to-the-minute tutorials on how to do various things with your system.
2. **Look it up at the LDP**. The Linux Documentation Project is an intense collection of free documentation on Linux, including the HOWTOs and several full-fledged Linux books (some of which are available from O'Reilly and others).
3. **Search netnews**. Usenet, or “netnews”, is a huge, ongoing discussion base on the Net, and the Linux newsgroups are among Usenet's busiest. Searching it with tools such as DejaNews and AltaVista and entering “Linux” and key terms that relate to your search will often yield positive results.
4. **Wade through Linux links**. The Linux Resources Page is hosted by the pro-Linux consultants at SSC (publisher of *Linux Journal*), and contains links to just about every Linux resource on the Net. Worth searching here are the archives to *Linux Gazette*, *Linux Journal*'s digital sister publication.
5. **Ask on IRC**. The #linux and #linuxhelp channels on IRC can provide you with instant help from a live person. Used sparingly, this can be an excellent resource for quickly finding out just what you need to know.
6. **Post on netnews**. If you still haven't found your answer, open up your news-reader software and post your question to the appropriate Linux newsgroup on Usenet. You'll be answered by a live human (probably several) who will help you for free. There's only one catch: once you too become a Linux guru, return the favor and spend a little time on Usenet helping out a newbie with a problem for which you know the solution.

Regional User Groups

Regional Linux User Groups have been popularized by the folks at SSC and have really taken off in the past six months. Called GLUE—Groups of Linux Users Everywhere—the idea of regional LUGs has spread to the point that there is now a LUG in almost every major city, and in many out-of-the-way places, too.

Talking to the folks at a LUG can be extremely helpful if you're a total newbie and need help with Linux. It's also a great way to see some of the amazing things that others are doing with Linux in your area—I always come out of our LUG meetings here in Cleveland having learned something new. And LUGs hosting Linux “Install Fests” are not uncommon; this is a meeting where you can bring in your computer, and LUG volunteers will install Linux on it for you for free (or a small donation).

Consultants

You may represent a company or other entity with specialized needs or requiring a great deal of support; if so, there is a whole *world* of Linux consultants out there. Start with the Linux-Consultants HOWTO (see Resources), but note that a number of traditional computer consulting firms and even ISPs now provide Linux consulting services.

Another rising trend is companies who provide preconfigured Linux systems. If you don't yet have the hardware, this can be a good way to purchase a running Linux system and not have to bother with installation.

Conclusion

These sources will almost certainly be able to fulfill any need you have in starting out with, or getting help on, a particular aspect of Linux. Yes, mastering Linux does require some effort, but the best things in life always do. That does not mean running Linux is any more difficult than non-free software or requires you to be a programmer—in fact, *anyone* can learn Linux! It's just that you do have to put some thought into the process, as you did when you learned to drive a car or to speak a foreign language. Those were worthwhile tasks, and you will find that the time spent learning Linux is, too.

Resources

Michael Stutz is a writer who frequently contributes to Wired News, on the Web at http://www.wired.com/. He is currently writing A GNU/Linux Cookbook for the Free Software Foundation, which describes how non-programmers can use GNU/Linux systems for their work. He can be reached via e-mail at stutz@dsl.org.

Archive Index Issue Table of Contents

Advanced search

Migrating to Linux, Part 1

Norman M. Jacobowitz

Issue #52, August 1998

Linux—not just for hackers anymore...

Linux? That's just for hackers and computer science students, right? Most experienced Linux users have heard that one a time or two. Over the past year or so, I have discovered that the answer is a resounding “no”—for Linux has something to offer everyone. You see, about one year ago, I took my first experimental plunge into the Linux world. I installed Linux on my home office computer, dual booting with MS Windows 95. Now, I run my home-based freelance business entirely from the Linux OS, using both free software and commercial products to get my work done.

This is the first in a series of three articles about my experiences migrating to Linux. I hope they will shed light on the techniques, methods, benefits and perils of using Linux as the primary OS in a small or home office (SOHO) environment. While these articles are written primarily for the non-technical end user who runs a very small business or works from home, hopefully everyone will have something to gain, regardless of their level of experience. In order to implement some of the specific suggestions contained herein, you should have a working Linux installation and a general understanding of Linux basics. If you are a complete novice interested in how Linux could potentially work for you, read on. Just one year ago, I was in your shoes. Now I'm running my business from Linux.

Why Migrate to Linux?

Just about every SOHO user would appreciate better performance from their OS. Yet switching to a different OS for your work is not a step to take lightly—you can expect a few growing pains. If you are in need of a better OS and are aware of the effort migrating may take, Linux is the first place to turn.

We'll now examine some of the pros and cons of Linux, based on the needs of a SOHO user. Personally, I'm looking for stability, ease of use and administration, low cost and a wide range of available productivity applications. Gaming, Internet and network services (other than dial-up PPP) and other capabilities don't fit into my equation. Your requirements, of course, may differ. Here's a more detailed view of each of these requirements.

Stability

What do SOHO users need from their OS? To put it simply, we've got work to do and no time for a flaky OS. We don't have time to reboot every half hour or to retype a document lost due to the failures of an inadequate operating system.

That's why Linux is so attractive. Linux is far less likely to suffer such frustrating failures. The underlying OS is more stable and the Linux development model sees to it that bugs are exterminated rapidly, with extreme prejudice. Commercial OS vendors have already sold you their wares and cashed your check by the time bugs are discovered. This means there is no pressing economic incentive to resolve difficulties on a timely basis.

As far as viruses are concerned, they are not a problem in Linux. The way Linux works, it is difficult, if not impossible, to propagate a virus. So-called “Trojan Horse” programs are still an issue, but viruses are not.

The combination of instability, viruses and bugs can make it impossible to get your work done while working in some of the mainstream commercial operating systems. Have you ever had MS Windows inexplicably freeze in the middle of a budget calculation? Have you ever lost two hours of work when a “General Protection Fault” crashes your word-processing session? I have had bad experiences ranging from mild frustration to major loss of income due to system crashes.

However, I have never—*not once*—lost data due to an OS failure in Linux. Well, once I completely hosed a perfectly good Linux setup, but that was my own fault. We'll discuss how to avoid and recover from such self-imposed disasters in the next installment.

Ease of Use

The first time the login prompt came up after my first Linux install, I was frozen like a deer in headlights. What, no GUI? No little “start here” buttons? Yet in just a matter of hours, I was cruising around the file system with no trouble and had the X Window System up and running. Next came the ISP hookup, and soon I was quite comfortable in Linux, learning more with each login session.

You may be saying, “That's great for you, but I expect quality support for my OS installation.” Veteran Linux users—as well as *InfoWorld Magazine*—know that the Linux community provides the best technical support currently available for any OS. Yes, I have had many problems with my Linux setup, most minor and some major. However, whenever I have posted a message to the Linux newsgroups, 90% of the time I have received several viable solutions and some good suggestions. Sometimes the answers come in just a few hours. This beats my experience with MS Windows NT by far (see sidebar). Even better, free Linux support does not expire, as does the free support under commercial operating systems.

Still, there is a long way to go towards making Linux more user friendly and a lot less scary. Yes, commercial GUI operating systems are prettier at first. Yes, on the surface they are a bit more intuitive. And yes, they are far less daunting in the beginning than Linux. Documentation and help systems for Linux applications are famous for being poor or nonexistent. That's why more and more Linux developers and vendors are rightly focusing their efforts on making Linux more friendly for the new user.

What I have discovered is that along with the very visible support from the Linux community, there is a “hidden” advantage to using Linux. To me, Linux exhibits a unique continuity to its learning curve. For example, I have spent several working hours learning to edit certain configuration files to my liking. Over time, I realized that the same methods often applied to editing files that pertain to a completely different part of the OS. When you first start to pick up on Linux, you may find each new piece of knowledge, each new skill, lends itself to easing the next task. In my experience, MS Windows does not share this continuity. Sure, the MS Windows learning curve is less steep, but there is a less reliable pattern to learning it.

In terms of usability, documentation and the quality of the user interface, Linux does need improvement. Still, the thousands of Linux developers worldwide are making dramatic progress towards putting Linux on par with many commercial operating systems in terms of usability.

Costs

How much does Linux cost? Of course, every introduction to Linux points out that Linux is free to everyone. It costs nothing, because no one “owns” Linux in the way Microsoft owns Windows or MS-DOS. To simplify things, a basic installation CD can be purchased from a vendor or retailer for $30 to $60 US. Plus, to get off the ground, one should really own one of the several fantastic books currently available for the novice Linux user for $20 to $60 US.

At the risk of being branded a heretic, I make the following assertion for the SOHO Linux user: using Linux is *more expensive* in the short term, but in the mid- to long-term, Linux provides a dramatic cost savings and rewards you with an attractive return on your investment of both money and time.

How can it be more expensive when it is free? Simple—the short-term opportunity costs are higher for the SOHO user who is currently using a commercial OS and is heavily invested in software made for that OS. This is one of the growing pains I mentioned earlier. For a cost comparison and a true story, see the sidebar. What's the moral to that story? For me, Linux is by no means free. I have spent hours learning to use Linux and tweaking my system when I could have been performing billable work. Remember, though, I have yet to lose a client's work or to be forced into redoing tedious tasks at my own expense while using Linux.

Plus, hardware can be used longer under Linux and quite often works much better as well. My aging Pentium 90 with 32MB of RAM is a bit sluggish when running current versions of MS Windows, but chugs along quite spryly under Linux. Linux has given new life to that computer, and I will be able to keep it up and running far longer than with MS Windows.

With all of these considerations, the economic value of Linux is clear. The mid to long-term return on investment more than pays for the short-term opportunity costs of switching to Linux from a commercial OS. Therefore, migrating to Linux is a sound business decision.

Available Applications

Right now, I run my business with a mix of commercial and free software. I am on my third Linux distribution, which I buy on CD due to the high opportunity costs of downloading large package files. I write direct marketing copy for my clients with a commercially licensed copy of Applixware. I dial my ISP using a dialing tool under the GPL. Right now my GUI environment is KDE. I send e-mail and browse the Web with the recently freed Netscape Communicator 4.04. All of these programs work as well, and often better, than their counterparts designed for MS Windows.

So far with Linux, I am able to do almost everything I used to do in MS Windows, with only a few high-profile exceptions. Right now I lack a good WYSIWYG page-layout tool like PageMaker or QuarkXPress, and I cannot get any of the freeware FAX managers to work for me as well as my commercial FAX program works under MS Windows. There are programs like WINE and Wabi that emulate MS Windows and allow you to run 16-bit MS Windows applications on Linux, but I have no experience using these emulators and cannot comment on their use.

What's the bottom line? Are there as many high-quality applications available for Linux as you will find for commercial operating systems? No. Is this situation changing every day, with more and more top quality freeware and Open Source software, as well as commercial Linux applications, released all the time? Emphatically, yes. Developing quality applications for Linux is a consuming passion for many developers worldwide, and that fact gives me the most hope for the future of Linux as a viable SOHO computing environment. With the level of excitement surrounding extremely high-quality programs like GIMP (GNU Image Manipulation Program), commercial vendors are learning they can profit handsomely from porting their products to Linux.

In short, migrating to Linux means more than just getting a stable OS. It means access to a collection of top-quality free and commercial applications, with more on the way each day.

One SOHO User's Experience

To summarize this first installment, I'll review the practical impressions I've gained during my migration from MS Windows to Linux:

1. Linux is more stable than commercial operating systems, eliminating the time and expense of redoing lost work.
2. Linux is more difficult to learn in the very short term, but has a better support system in the long run.
3. Linux—while “free” in the absolute sense—brings with it a host of opportunity costs in the short term but pays for itself many times over in the long term.
4. Applications for Linux are growing in number and quality each day.

Now we have a basis for a continued and more detailed exploration of migrating to Linux from a commercial OS for the SOHO user. In the next installment, I'll get into the details of migrating, discussing such vital matters as basic system administration and making reliable backups. I'll get more specific about some of the software that is available for Linux and compare it to software available in the commercial OS world. I'll also discuss how to interface and share work with our colleagues and clients who are stuck with their commercial operating systems.

Comparing Costs for a SOHO Setup: Red Hat Linux 5.0 vs. Windows NT

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

Archive Index Issue Table of Contents

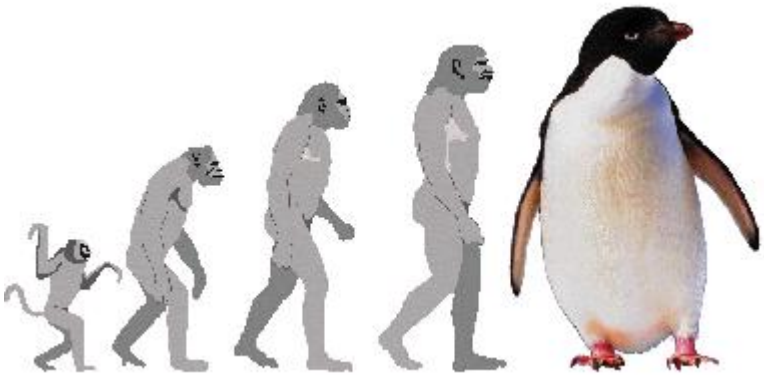
Advanced search

Betting on Darwin

Doc Searls

Issue #52, August 1998

Doc Searls interviews Marc Andreessen and Tom Paquin on Netscape's Open Source Strategy.



> Evolution is not a force but a process; not a cause but a law. It is not enough to do good; one must do it the right way. No man can climb out beyond the limitations of his own character. —John, Viscount Morley of Blackburn

Only two operating systems are growing in market share today. The one we all know is Microsoft Windows (both NT and 95). The one most of us don't know is Linux. After reading the mainstream press (or listening to politicians like Orrin Hatch or to professional haranguers like Ralph Nader and Gary Reback), it seems clear that Microsoft, that evil monopoly, is making Windows the requisite operating system for every computer-like device in the known universe.

But there is a constituency that won't let this happen. It's the growing community of hard-core technologists, an increasing number of whom now rely on Linux. A case could be made that the only technologists who aren't into Linux remain with another OS for practical or business reasons.

Today, Linux is the technologists' OS. It's a bad idea to bet against them. It's a good idea to bet with them. That's exactly what Netscape did when they released their source code to what everyone suddenly calls the Open Source community, but for years was the Free Software movement. Significantly, Linux is free. But more significantly, Linux's source code is open. Anybody can look at it, tweak it and share their tweaks with friends—which is exactly why Linux is so popular with techies. Quite literally, it's theirs. They made it.

“All the significant trends start with technologists,” Marc Andreessen says.

Lest we forget, the Internet was created by technologists, and its explosive growth is far more an expression of rampant hackery than of commercial activity, personal expression, massive archiving or whatever.

Consider this: over half of all the web pages on the Internet today are produced by free web software—mostly Apache—running on Linux operating systems. Windows NT is gaining, but not necessarily at Linux's expense. In fact, Linux is gaining too, mostly at the expense of commercial systems.

Even many commercial sites use Linux. According to a Commerce Department report released today, the volume of information being processed over the Web is doubling every sixty days. The mind boggles.

If even half of that growth is happening on Linux, the population of supportive techies is at critical mass or better. Netscape knows this, which is why they're betting the farm on Linux, just like they bet the farm on the Net four years earlier, when they changed the whole software business forever by freely releasing their first browser to anyone in the world with an FTP client, and by making development partners out of a ubiquitous user base.

In fact, the farm metaphor is an apt one, because releasing the browser source code on March 31 amounts to a bet that spring is here, the ground is tilled, the seeds will grow, and the crop will be abundant. It's probably not a coincidence that Marc Andreessen grew up on a farm.

He knew in his bones that releasing the Netscape browser source was the “Right Thing” to do, so did many other farmers out there in the hacker community. Chief among these was Eric Raymond, who earned some notoriety when he edited *The New Hacker's Dictionary* a couple of years ago, but whose fame really caught fire when his essay “The Cathedral and the Bazaar” (http://sagan.earthspace.net/esr/writings/cathedral-bazaar/cathedral-bazaar.html) caught the attention of the Netscape folks, and spread from there.

By Eric's metaphor, significant software was traditionally built like a cathedral, “carefully crafted by individual wizards or small bands of magicians working in splendid isolation, with no beta to be released before its time.” But a new model was brought to the world by Linus Torvalds, the creator of Linux: “release early and often, delegate everything you can, be open to the point of promiscuity.” The result was a vast community of techies that “seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, which would take submissions from anyone), out of which a coherent and stable system could seemingly emerge only by a succession of miracles.”

The cathedral and bazaar metaphors seemed apt for the worlds that surrounded the code. The corporate world was formal and contained, while the open source world was informal and open.

Not that the bazaar lacks formalities of its own. (A detailed analysis of these formalities can be found in “Homesteading the Noosphere” by Eric Raymond, http://www.catb.org/~esr/writings/homesteading/.) Every society has its protocols, and the hacker community is no exception. The open source world is a society of peers. “You're not a hacker until somebody else calls you one,” Eric says. What you get out of this society is not just great code written by the best programmers in the world, but massive peer review and an equally abundant source of support.

I found myself pulled into this world when (*Linux Journal* publisher) Phil Hughes told me I ought to be talking with Eric, since for years I've been a kind of Johnny Talkyseed, spreading the notion that markets are conversations, rather than battlefields. Phil thought the bazaar metaphor was at worst a corollary and at best another way of saying the same thing.

Over the next month, I got a lot of great hang-time with Eric (he was a house guest for a week). I also intercepted Jim Barksdale at an industry forum and turned that conversation into an interview with Netscape co-founder Marc Andreessen and Mozilla.org's new captain, Tom Paquin. I attended the Silicon Valley User's Group (SVLUG) meeting where Marc and Tom together announced the open source move and addressed the Linux community on their thinking for the first time.

In brief, here's what Marc and Tom told that community:

- The timing is right, because both open source and Linux energy have reached critical mass.
- Releasing the browser source and joining the open source community is a return to roots, both for Netscape and for Marc personally.

- There's an opportunity to catalyze open source development, not only in the Linux community, but inside two cathedrals: commercial software developers and their corporate customers.
- If Sun doesn't open Java's source code, it'll happen anyway, outside Sun's cathedral. The open source community should work to bring Sun to a “more enlightened” place.
- Sun, Oracle and other mainstream developers need to adopt Linux as a Tier One reference platform. Netscape already has.
- Linux has a couple of problems its development community should address. One is scalability. The other is mission-criticalness. In time, it should be able to run on multi-processing servers and clusters and handle heavy-duty transaction processing, for example.

Marc also made some predictions:

- Linux will get discovered by the mainstream press within a year.
- Linux will consolidate the UNIX market and expand beyond it.
- Netscape will bundle Mozilla with Linux.
- Mozilla will become the GUI of the future—“the user environment where people can live and work on the Net.” Here it will naturally go “hand in hand, or hand in claw, with Linux.”
- When mainstream vendors want to start adapting Linux to their particular hardware, they're going to run up against the Linux GPL. Something needs to be done about that, or Linux adoption will be slowed down.

A few days later I was booked for a half-hour interview with Marc and Tom at Netscape. The conversation quickly reached critical mass and ran for an hour and a half. What it comes down to, it's clear, is one line from Tom: “We're betting on Darwin here.” Let's hope he's right.

The Technologists and their OS

Doc: Why release the browser source now?

Marc: Today we're at an inflection point, a trigger point, when there's an alignment with the energy of growth. Linux is hot. The technologists have adopted it, and it's growing fast all through the open source community. This gives us confidence that we couldn't screw it up if we tried. And now we can't de-commit. This thing has been set in motion. Even if we do nothing, there's no stopping what's going to happen. It's going to be fun.

Doc: A few years back, somebody at Netscape called—maybe it was you—and said, without meaning insult, that the OS is “just a device driver.” Is Linux well suited to that role?

Marc: Yes. Linux is evolving naturally to do what an OS does best, while Windows is constantly adding non-OS functionalities as part of a vertical strategy to take over layer after layer above the OS. This looks a lot like what IBM did first and Apple did all over again on the IBM model. It has made Microsoft much more vulnerable to a grass-roots technology movement like Linux, and is one of the reasons Linux is the only non-Microsoft OS that's gaining market share. It's amazing that these companies make the same mistakes over and over again, but they do.

Doc: There is an irony here. One of the reasons why PCs rather than Macintoshes were established in companies, was that, although everybody agreed Macintoshes were better, the technologists would rather work with DOS because they could easily write DOS programs that addressed a specific problem. In the same way today, the technologists would rather work with Linux. What you said at SVLUG about customers using Macintoshes and PCs at work and Linux boxes at home isn't just a joke. There's something going on here.

Marc: Absolutely. Technologists are driving progress, and it's easier to drive with Linux than with anything else.

Doc: Tell me more about your role in the Linux movement.

Tom: What we want to do is give Linux critical mass so it hits a crossover point. There's a huge development community that's growing bigger every day. And it's an efficient one relying on massive peer review that you can't replicate inside the corporate world, although the two overlap a great deal. So we're betting on Darwin here—natural selection. All we're doing with Mozilla.org is helping create conditions that give evolution a hand.

Marc: It's the exact same thing I saw with the Net. Before it hit critical mass, the Internet had a million plus users. People were using it every day, and all of them were technologists. But the Net was also unapproachable and not understandable to people outside that community. They'd scratch their heads and wouldn't get the point. The crossover point came with e-mail and the Web —though I think e-mail was the bigger driver. Suddenly it became relevant to everybody. But first it had to reach a critical mass in the technologist community. Linux is there today, and it's ready to cross over. But it requires specific catalyzing events, just like the Internet did.

Doc: And releasing the Mozilla source may be just the event.

Marc: Right.

Getting to Scalability

Doc: You say you are looking for Linux to obtain a scalability to support a lot of database transactions and large-scale information processing that Solaris, AIX and those other UNIX variants perform. It seems to me the kind of guys who hang around Linux are not the kind of guys who want to do that kind of programming. Is that true?

Tom: You mean put SMP (symmetric multi-processing) into the kernel?

Marc: Right.

Tom: All you need is a couple or a few people who can steer and many other people who say “Oh, that's an issue? I'll deal with it.” And competent programmers will be all over it.

Marc: Especially since this has already been done multiple times.

Tom: That's my point. This is not uncharted water. Companies have done it. They know where the problems are.

Doc: But it's uncharted water outside of companies, and Linux is sort of this extra-corporate thing. Does the Open Source community sit around caring about this sort of issue?

Tom: If you asked this question six or seven years ago, then you'd have a hard time finding a large and broad enough variety of people who have spent time making, for example, SMP kernels work. In 1998, that's no longer true. You can scoot up and down the street and find people who have done it. A lot of those guys go to the Linux Users Group meetings and are interested in getting SMP in the machine they've got at home. It's just not that remote of a science anymore. So I think your characterization is no longer valid. Marc, do you know Linus's position on this? He thinks it's time to do it, right?

Marc: Right.

Doc: So then it's done. It's going to happen.

Tom: That's right.

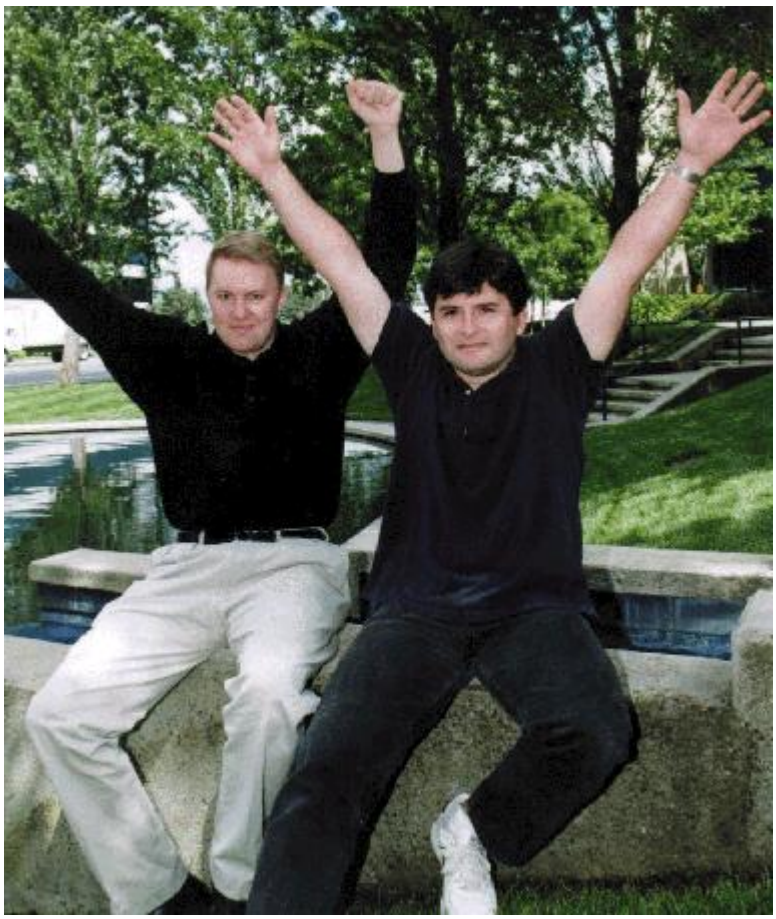
Doc: Would you dedicate any internal resources to making it happen?

Marc: Not currently. If there was something specific we could do to help, maybe. But I'm not suggesting Netscape should take more of a leadership role

in determining the future of Linux. That seems to be working just fine. Of course, in any case where we can help, we'd be interested in figuring out how.

Doc: I'm trying to figure out how this larger community works. Some of these guys are working at this on their day jobs and some aren't.

Marc: Typically they have very relevant day jobs. At least one person in every technology company on the planet is in this community; so again, this is the same thing that happened with the Internet, and why it broke into the mainstream. It infiltrated itself into all these companies through resident technologists who were interested in it.



Marc Andreessen and Tom Paquin believe in the future of Linux “this much”.

From Authors to Stewards

Doc: What's your role in the browser conversation now, with Mozilla.org?

Tom: We're hoping to steward that conversation, but there's no compulsion here. They have the source and can choose to go their merry way. We're all better off if there's a lot of coordination and cooperation, and the community recognizes that. So it's not that they're going to be dragged kicking and screaming into it. We don't have to offer free donuts to get them to come to the party. On the other hand, if they don't like our stewardship, they don't have to

cooperate. So we have to act in the traditional open source responsible manner—in a way that will make them want to play. Somebody will steward this, whether it's us or not. If they don't like us, they'll find someone else to do it.

Marc: Again, we can't de-commit.

Doc: I've always thought the best kind of marketing amounts to arson. You set fires and stand back and watch what happens.

Marc: (Laughing) Right. I'll enjoy watching this one.

Doc: The open source world, it seems to me, is already a very active conversation. Great kindling, as it were. Already the browser conversation has changed: Whoosh, it's different.

Marc: Right.

Doc: Yet the press will still play this as a war over a battlefield, no matter how much it looks like the Big Bang, and no matter how non-territorial it is.

Marc: Exactly. What's really happening is new vendors constantly opening up new spaces where new things happen. Everybody gains. That's the real story.

Doc: And, once you have a firestorm of interest going on around Mozilla, it will do what LDAP did two years ago.

Marc: Sure. Same pattern.

No Surprises

Doc: What has surprised you, in just the week since the source went out?

Marc: A lot of *our* programmers have been surprised by the level of interest and activity. We've cured a lot of skepticism in the last week.

Tom: You're going to dislike the things that surprised me, because in truth we expected this to take off in exactly the way it has. We got a bunch of memory bug fixes that I thought were great. I was a little surprised at how strong the Macintosh community jumped on. They were really in there. We got a localized version that happened fast: a crypto plug-in from the crypto-weenies in Australia who hacked it together in seven hours hacking time, fifteen hours real time. Speed was impressive too. I expected people to spend more time digesting, but they piled right in.

The skeptics are surprised, of course. “Well, gee, we see some evidence here that Netscape is serious about this and not abandoning the browser business.” *That* kind of stuff is what's truly surprising.

Marc: There was skepticism in engineering as to whether people would be able to understand it enough to make changes. And they've absolutely been able to do so. It hasn't been an issue.

Tom: Yeah, the difficulties haven't been difficult.

Hand in Claw

Doc: Marc, you said at the SVLUG meeting that you could see Mozilla becoming the GUI for Linux. What are you talking about there?

Marc: I don't want to overreach and say Mozilla has to be *the* GUI for Linux. It makes perfect sense for Mozilla to be in a window running on an X desktop or anywhere else. But the thing we think is happening—and has been steadily happening over the last five years—is that we are all spending more and more of our time doing things on the Net as opposed to doing things on the local system. That should lead to a major interface change we're only beginning to see, equivalent to the shift from text to GUI.

The next shift should be from a GUI-centric or desktop-centered interface to a net-centered interface, one that takes into account the virtues of the network. Because the fundamental difference is that there's a million or a billion times as much stuff out there that we have access to and therefore may have to deal with at some level, than was ever on an individual desktop system. And so, the interface has to change radically to whatever the Net opens up. The scope of that means Mozilla is positioned to become that GUI, that breakthrough, through community effort.

Doc: The network interface?

Marc: Exactly, the network interface. If you think of Mozilla as a full-screen environment that is inherently a network interface, it is heavily oriented to filtering and managing the huge amount of information out there, and being extraordinarily personalized for the individual user. There are all these resources to draw out of the Net, which increasingly represent the needs of the user to the Net; thus, resources become smarter in a sense, doing more useful stuff for the user, and finally will, no doubt, include the contents of the user's own machine.

Doc: So your local machine becomes a subset of what's in the networked universe, but your view is through a Net-oriented interface.

Marc: Exactly. The local box is a very, very small subset, but the Net is the context.

Doc: We already see people adapting the way they work to the Net context.

Marc: Now, we see lots of documents, reports and messages are being typed in e-mail that previously would have been typed in a word processor or on a typewriter. You see the emergence of the navigation center in the code that's up there now, providing a view of the Net. Maps of web sites, plus bookmarks, plus directories, plus e-mail messages—basically everything you have access to, in a single place.

I can easily imagine Mozilla, within a year or two or even sooner, will have an interface mode that gives you sort of a full-screen experience that happens to provide everything you need to manage your local desktop but is specifically focused on being a very effective interface for the Net. This is going to be a very fertile breeding ground for a lot of innovation in user interfaces. There has been a huge amount of work in computer science labs and in the larger community on next generation user interfaces. Historically, it has taken decades for any of those innovations to make it out. Windows and the Macintosh are known for technologies invented 25 years ago. But while it takes a long time for this to happen in the commercial marketplace, it can happen much faster in the grass roots world that will grow around Mozilla. You will see grad students doing radically advanced user interfaces as their Ph.D. theses, implemented in Mozilla. One of those will be the interface we'll all be using a few years from now.

Doc: I have this notion that who you are is more a matter of where you come from than any other factor. It's the anchor point for the vector of your life. You can change your name, your job, your whole résumé, but you still come from the same place. And this is true of companies as well as people. It's the source of character. Apple will always come from Steve Jobs' aesthetic. Sun will always come from The Network. Netscape will always come from wherever you, Marc, were at when your team created Mosaic. Was that UNIX?

Marc: There were UNIX guys, but a lot of PC and Macintosh guys as well. Essentially, where we came from was a commitment to a heterogeneous universe. We also go back a long way with this. We had connections in the open source community before Linux really existed. Back then, the open source community was a lot smaller. The projects they were focused on were things like Emacs, FreeBSD, C compilers and so on. Now that community is larger, more involved with the Web and growing very fast. That's why the timing is so right for this move. It wouldn't have been a few years ago.

Social Computing

Doc: A while back it seemed to me that the next stage beyond personal computing would be social computing. It seemed a natural progression, from the one to the many. But there is a difference in kind between the personal and the social, between the user interface to a personal computer and the user interface to the Net, which is where we find computing's society. What you're talking about is making Mozilla a social interface.

Marc: Right.

Doc: Maybe it's a stretch, but the Open Source society is very different than the company that's trying to build the ultimate personal computer.

Marc: Right. Microsoft is trying to put all this stuff back in the box, right? They're trying to take this whole world and squeeze it down.

Doc: If I look for analogous concerns in the real world, my most personal space is maybe my closet, because I know where my shoes are, and my shirts and belts and so forth.

Marc: Same for your bookshelf.

Doc: Yet, society is nothing like my closet or my bookshelf. But the presence of a computing society in my life means I don't write in a word processor anymore. I write in e-mail and a text editor. I wrote my questions today in BBEdit, saved it to the a42 server at *Linux Journal* in Seattle, reviewed them with Phil Hughes, wherever he was, and printed it out in my office. The browser was there for all of it, of course. And the interesting thing is that all this is far less

feature-rich than what I used to do on a word processor and print out at home. But it's far more social, and far more useful.

Marc: It's social publishing.

Doc: Right. Now, here's where I'm going with this: we are each willing to yield a lot of personal choice to get along with society. Hey, maybe I like to race cars, but I won't do that on a highway. But I behave the way I do on a highway because that's a social place.

Tom: That's incredibly true of your social life. All of your manners are necessary to get along in society.

Doc: The irony of personal computing is that you can't see the social from the personal. I can't abstract the organization of the world from the contents of my closet. I can't understand traffic from the perspective of racing a car. I can't see more in terms of less. Yet there are concerns and functions inherent in social computing that don't show up in personal computing. What I'm suggesting is that you guys live at the social level and have lived there all along.

Marc: That's right.

Doc: So you're coming from the social, and Microsoft is coming from the personal. Which is why I think you're not surprised, Tom, when the society you know best acts just as you expected when you released the source code.

Tom: Right. And somebody who lived in the other world might be asking all these questions about, “Why would these people want to help you?”

Marc: Or “Why isn't this just going to fragment?” That was the big question we got from the press, and we had to carefully explain why there are tons of reasons it won't fragment, not the least of which is the centrifugal force where, if you want to fragment, you take upon yourself the burden of pulling in all the changes everyone else is making into your own version. There are always issues like this that actually make things work. That's why Linux works.

Tom: Who wants incompatibility? The problem is, I could say this to a room full of people and they'd all have the same answer. The guys up north want to sell tools that are the only tools in the business that can operate on the data... Then I say “Ah, okay, sorry, I wasn't thinking about that. I think compatibility and open interfaces are good things.” So we don't communicate because we're not operating at the same level.

Dealing with Java

Doc: Is Sun going to open Java?

Marc: There are eerie parallels here between what Sun did in 1982 and what Linux is doing in 1998. In many ways it's the same technical community. It's a community that's very focused on free versions of UNIX, running on commodity hardware, appealing initially to a very technical audience and eventually with a much broader relevance. Sun, in 1983 and 1984, totally and uniquely had an understanding that you could harness all this energy around BSD and commodity hardware and just get it out the door to this technical community. SGI and many other companies didn't get it back then, but Sun did.

Now they've done a complete 180. If you think about it, they are now a proprietary verticalized systems vendor: doing their own chip, their own complete systems architecture, their own software, their own operating system, their own applications in many cases, their own storage devices and so on. When they think of Java, they think, “we really need to control this.” Wrong answer. Because the alternative is that there will absolutely be multiple implementations. There need to be Javas defined to lead naturally in the open direction. Sun is going to force them to emerge and to flourish by not opening Java up.

Tom: It's like what AT&T did in the UNIX market.

Marc: Right. If you think about it, what's so 180 is—well, look—I like Alan Baritz a lot. But Alan's from IBM. In 1982, Sun would not have put someone from IBM in charge of something like this. Now they would, and that's a mind-set change. And so that's one reason this is so fascinating. They play a very strong marketing battle on the one hand about why Java is open, and on the other hand, they're not open at all.

For our part, we can make a practical observation that if we can't build Java source code into Mozilla, there will be a Java implementation that will end up in Mozilla. It'll be Sun's or someone else's.

Tom, should I pre-announce something here?

Tom: Go ahead.

Marc: We are at work on a next-generation Java runtime, internally code-named Electrical Fire. It should be on the Net shortly. There are multiple efforts already launched here. This is not a complete Java runtime in and of itself, but it's technology around which to build a very high-performance Java runtime. And we're going to put it on under MPL (the Mozilla license).

Natural Selection

Doc: Are engineers starting to beat your door down to work here? One of my hacker friends wants to know that.

Tom: We're seeing fewer people saying “Hire me, hire me” than those who are saying, “I want to play.”

Marc: We're hearing from people who want to do some particular project and ask if Netscape would be willing to subsidize it. And we could potentially use some number of those things, but I'm not sure it would be that good for Netscape to try to fund some of this stuff because it gives the wrong impression and offers the wrong incentives at that point. But I don't know. It's complicated. Also, from a practical standpoint, there's a limit to how much we can fund.

Tom: I like the Darwinian approach. Let natural selection occur.

Doc: Say, “If you've got such a good idea, go get your own funding. Start a business.”

What is this going to do to the server side of your business?

Marc: Indirectly, it benefits. Let me explain why. The server side of the business is strong and healthy to the extent that there is a competitive client marketplace. If there were a noncompetitive client marketplace and Microsoft had 100% client penetration, we would have a difficult time in the server market for obvious reasons: Microsoft would use its advantages. But now, because there is a competitive client marketplace, not only does Microsoft lack a dominant share, or even a majority share, they are forced to implement open interfaces and open ABIs, such that our servers actually interoperate with their clients. They are forced to do that by the market. As long as there is a competitive client market, their customers will demand it or they won't stay customers. If that market persists, our server business does great. If that market collapsed down to where Microsoft is the only competitor, our server business—and everyone's server business—is in a lot of trouble. That's why, for example, we're not going to try to get features built into Mozilla that only work on our servers. Open browsers guarantee an open server market.

Doc: Do you have any concern that this Open Source community will go create free server software that competes with your commercial server software? Apache equivalents for security, management and the rest of it?

Marc: Those exist now, in most cases. Freely available mail servers, security solutions, Kerberos, directory systems, University of Michigan LDAP. We're

comfortable competing with all that. We sell $75 million worth of web servers every year, in a market where both Apache and IIS are free.

Doc: And you've got Caldera bundling your stuff with Linux, and there's a market for that.

Marc: Sure.

Doc: So you're saying that the world is exploding at a sufficiently rapid rate that it's going to suck your stuff into it at least as fast as anyone else's.

Marc: We'll sell more web servers this time next year in spite of the fact that more free ones will be out there. We sell software to businesses. Individuals are fundamentally looking for a different set of things.

What's the Object (Model)?

Doc: You've been trying to establish an object model built around IIOP (Internet Inter-ORB Protocol) and CORBA (Common Object Request Broker Architecture), versus Microsoft's DCOM (Distributed Common Object Model). Does this Open Source release play in that at all?

Marc: Good question. Tom, we took the ORB (Object Request Broker) out, didn't we?

Tom: I don't think it's there.

Marc: That's interesting. [We] made the client ORB-free in source form.

Tom: It's going to have an ORB; I can't tell you exactly when. I don't think the world is going to tolerate a vacuum in this space for very long. I've seen a few proposals. All the existing popular options have some kind of fatal flaw in the eyes of part of the community. The DCOM guys have a problem with this part of an ORB. And this CORBA group has a fundamental problem with the portability or interoperability of some part of DCOM. There are just wave after wave of these concerns. We've had arguments within Netscape about the best way to address the question. I'm looking for Darwin taking over on this one, too. I think it might be a spirited debate, and educational for me. I am not an objects guy.

Doc: I was talking to Phil Hughes about this and he said the same thing. But he added that it all depends on your generation—when you grew up.

Tom: It's true.

Doc: He said, if you go to the schools now, everybody's talking objects. Maybe not so for the older guys.

Tom: Some of those older guys have evolved into objects people. Their arguments are very interesting. I follow them, but I can't generate them. In any case, I think it's going to be interesting to get this out in the open.

Marc: I bet we'll have native IIOP built into Mozilla in source code form within a year. From somewhere.

Doc: Meanwhile, there seems to be faith that the world is going to be made of objects, and there will be an enormous conversation in objects between clients and servers all over the world. I don't know to what extent evolution toward that state is constrained by the presence of two rival object models.

Marc: As an issue, it's not pragmatically useful or important to enough people yet. The results may be what you describe, but the object people get confused between the way the world's going to look in N years versus what it's actually going to take for interest to pick up and carry us there. This is why it was so hard to time the adoption of LANs. In retrospect, it took time for all this technology to connect before it made more sense to print over the LAN as opposed to carrying floppy disks. To get to the current state of the Internet, we had to get technology to a point where folks wanted e-mail accounts in order to write to their kids in college. I think this applies to programmers too. What's the trip-over point where all of a sudden it really matters? Personally, I think it's going to happen when people build more sophisticated multi-tier applications. Then they're going to need smarter communication between clients and servers than they're able to get today. And they're already going to be developing a lot of their logic on the server in the form of objects because they're using C++ or Java, and also off the client. And so, they're just naturally going to start to interconnect those two.

Doc: Directory service is going to play in a really huge way here. You've got a directory server.

Marc: A damn good one.

Doc: That seems to be optimized for this. Would you tweak it to match what's going on in the outside world? Let's say you put the ORB in the browser, watch what happens, see who's trafficking in objects, how it looks and who needs a directory.

Marc: Absolutely. A lot of what we do in the servers will be influenced by what happens with Mozilla. It's sort of always been that way. Fundamentally, there's

a chicken and egg problem with some of these things that involve all kinds of servers, and it usually requires you to get some amount of user action or adoption going on before lots of stuff starts to happen on the server. I'll bet that's exactly what's going to happen in this case.

The Hot Dog Stand

Doc: I like your faith. But your critics have said "It's a desperate move, and they didn't have any other choice."

Tom: That's driving me crazy. Any business decision is going to look vague when the costs and risks versus the opportunities and benefits are highly balanced. The teeter-totter teeters. In this case, the cost of releasing the source to the Navigator plummeted in the last nine months or so. Now, it's a no-brainer—time to pursue a new opportunity.

Marc: The flip-over point was when we made the binaries free. After that, releasing the source made perfect sense.

Tom: The benefits and opportunities have always been strong. It's just that the costs have been high. Well, they have not always been strong. I have to be fair. In 1994, there weren't many developers out there who "got" the Web—whose heads operated in that space. And if you invited everybody to come and play, the signal-to-noise ratio would have been zero. Since then everybody has come to know about the Web, and they have attitude and interest and ideas, so for some time the opportunities and benefits have been on the strong side. But when the costs plummeted, then suddenly it was a lot easier decision to make, regardless.

Marc: Those statements usually come from people who are typically not involved in businesses. Or at least not businesses of scope. There's a natural decision-making process businesses go through to try to maximize their economic opportunity.

Doc: I liked the way Jim Barksdale put it when he said, "We had to get revenues low enough, so we could make this choice."

Marc: Right! We had to walk it all the way down to zero. Along the way, we used that revenue to subsidize all these other products and services that now generate revenue.

Doc: Change goes on all the time anyway. All products have a life. Either they change completely in order to live, or they die.

Marc: Yep.

Doc: That's true of Windows, of browsers, of all kinds of things.

Marc: Which is why it's ridiculous to criticize a company for changing.

Tom: The criticism is, we're in power and we have competitors, so we've got to be desperate to give all this power to our competitors.

Marc: To some people it's “Aw, they're desperate.” And it's obvious they haven't even thought it through any more than that. It's a pointless comment. Jim Barksdale abbreviates this to “They haven't ever run a hot dog stand. What do they know?”

Doc: The problem with the press is that when you're running a sportscast, you have to fill the air with sounds of competition even when there's nothing going on.

Tom: You're really right about them using war terminology, using that chest of war words, and when things don't fit that model, the brains break. The stunning thing is their unwillingness to put that box away and break out a new box.

Marc: It's especially interesting when you think about it, because there are so many different sources of media now—so many different voices out there, striving for unique coverage and unique angles. You've got an inherent unwillingness by a large number of them to actually do anything unique.

Doc: Part of it is time pressure: you have to say something. Another is the story principle. Stories are the fundamental unit of consciousness. Conflict is interesting, and most stories are about conflict. No story starts with “And they lived happily ever after.” What keeps them turning pages is the conflict that's going on, and it will still be going on after they turn the page. That is inherently interesting to human beings. The Great Harmony is not interesting. A bunch of people hacking on something for the pure fun of it and for peer review is not as interesting to the press as the “Great Browser War”.

Marc: Right.

Doc: Eric Raymond told me today that you're not a hacker unless somebody else says you're a hacker.

Marc: That's true. He's absolutely right.

Doc: There is also a peerage among the Dilberts of the world that is invisible to those further up the corporate hierarchy.

Marc: That's true.

Doc: So there is all this interesting stuff going on, but it doesn't involve conflict so it isn't that interesting to the press.

Marc: Now there are more people in the technologist community than there ever have been. More people are comfortable with technology now at every level than there were five to ten years ago—just a huge number more.

Doc: Anything else you want to add before we wrap this up?

Marc: I think over the next six months we'll have an explosion of activity. What pops out of this new system will be interesting. You can view it as an amazingly complex system or organism or computational device on a large scale, with its own set of rules and organizing principles. Stuff is bound to pop out. By definition, what pops out will be the right set of stuff. It will be self-fulfilling.

Doc: This brings to mind John Seemly Brown's matrix, where he puts social computing in context. He says there are some things that only we know. They emerge in the social space, rather than the personal. And most of the shared knowledge is tacit rather than explicit. Tapping into this tacit dimension is what you have in mind, I think.

Marc: Absolutely.

Tom: Domestically, among the people here at Netscape, I've been explicit, saying, “Look, I'm not going to explain to the Open Source world how open source works. Build it; they will come.” If they need to be sold, they're too much work for me anyway. As long as I believe there is a large community out there—that will come, based on tacit communication—I'm in business. And the rest of the people can learn how this thing works. For some people, however, I have to be clear: “I'm expecting this behavior.” This allowed Mozilla to say, “We're just kind of doing this,” and expect Open Source behavior to run with it. Yeah, absolutely. At the tacit level, the Open Source people totally get what's going on.

Doc: In a way, you're saying “We have built it, now they'll come and rebuild it.”

Tom: Oh yeah. We build the place where source lives, and give them an opportunity to get in there and get their fingers dirty, and no one will be able to keep them away.

Doc: Like a free Home Depot for developers.

Tom: Yeah!

Marc: Right. We've got a few things we think we can stock it with on top of Mozilla. I'd like to see what the community does with Electrical Fire.

Tom: The list of things like Electrical Fire is about seven, eight, nine, ten items long, all of which are majorly interesting. So, we've got some stuff. We can set some fires.

Resources

Doc Searls (searls@batnet.com) is President of The Searls Group, a Silicon Valley consultancy, and a co-founder of Hodskins Simone and Searls. He has been writing on technology and other issues for most of his life. The *Flack Jacket* series of essays is collected in *Reality 2.0*, http://www.batnet.com/searls/docworks.html. Other series are *Positions* and *Milleniana*.

Archive Index Issue Table of Contents

Advanced search

Selecting a Linux Distribution

Phil Hughes

Issue #52, August 1998

Having trouble deciding which distribution to go for? Here's help.

Any current Linux distribution most likely contains the software needed to do your job, including kernel and drivers, libraries, utilities and applications programs. Still, one of the most common questions I hear is “which distribution should I get?” This question is answered by an assortment of people, each proclaiming their favorite distribution is better than all the rest.

My new theory is that most people favor the first distribution they successfully installed. Or, if they had problems with the first, they favor the next distribution they install which addresses the problems of the first.

Let's use me as an example. SLS was my first Linux installation. Unfortunately, SLS had a few bugs—in both the installation and the running system. This, of course, isn't a surprise since this installation took place five years ago.

Now, about this time, Patrick Volkerding came along and created Slackware. Pat took the SLS distribution and fixed some problems. The result looked the same as SLS and worked the same, but without bugs. To this day, I find Slackware the easiest distribution to install.

I have, however, progressed beyond installation problems and found some serious shortcomings in Slackware which have been addressed by other distributions. Before I get into specifics, here is a rough estimate of the number of times I have installed various distributions, in order of first installation. I give you this information to help you understand the basis of my opinions.

- 100+ SLS/Slackware
- 5 MCC (a small distribution done for university students)
- 5 Yggdrasil

- 20 Red Hat
- 10 Caldera
- 20 Debian
- 5 S.u.S.E.

That said, here is my blow-by-blow analysis of what is right and wrong with each distribution. Note that this is my personal opinion—your mileage will vary.

SLS/Slackware/MCC

All these distributions are easy to install and understand. They were all designed to install from floppy disk, and packages were in floppy-sized chunks. At one time, I could successfully install Slackware without even having a monitor on the computer.

There are, however, costs associated with this simplicity. Software is saved in compressed tar files. There is no information within the distribution that shows how files interrelate, no dependencies and no good path for upgrades. Not a problem if you just want to try something, but for a multi-computer shop with long-term plans, this initial simplicity can have unforeseen costs in the long run.

Yggdrasil

Yggdrasil offered the most promise with a GUI-based configuration. Unfortunately, development stopped (or at least vanished from the public eye), and it no longer offers anything vaguely current.

Red Hat

When I first looked at Marc Ewing's creation, I was impressed. It had some GUI-based configuration tools and showed a lot of promise. Over the years, Red Hat has continued to evolve and is easy to install and configure. Red Hat introduced the RPM packaging system that offers dependencies to help ensure loaded applications work with each other and updating is easy. RPMs also offer pre- and post-install and remove scripts which appear to be underutilized.

Version 4.2 has proven to be quite stable. The current release is 5.0, and a 5.1 release with bug fixes is expected to again produce a stable product.

The install sequence is streamlined to make it easy to do a standard install. I see two things missing that, while making the install appear easier, detract from what is actually needed:

1. The ability to save the desired configuration to floppy disk during the installation process (something that both Caldera and S.u.S.E. offer) would simplify subsequent installations on the same or other machines.
2. The ability to create a boot floppy disk during installation.

Red Hat has evolved into the most “retailed” distribution. First it was in books by O'Reilly, then MacMillan and now IDG Books Worldwide. It also appears to have a large retail shrink-wrap distribution in the U.S.

Versions of Red Hat are available for Digital Alpha and SunSPARC, as well as Intel.

Caldera

The Caldera distribution was assembled by the Linux Support Team (LST) in Germany—now a part of Caldera. Caldera, like Red Hat, uses the RPM package format. Installation is similar to Red Hat with the addition of the configuration save/restore option.

Caldera is different from other distributions at this time in that it offers a series of systems including various commercial packages such as a secure web server and an office suite. Caldera is also the most “commercial feeling” as far as packaging and presentation.

One complaint I received from a reviewer of my original version of this article is that you cannot perform an upgrade. That is, you must save your configuration files and then re-install.

Debian

Debian is one of the oldest distributions, but because development is strictly by a team of volunteers, it has tended to evolve more slowly. Since development is performed by a geographically diverse group, the ability to manage and integrate upgrades is of primary importance. To that end, you can always upgrade a system by pointing it at an FTP site and instructing it to get the latest versions of all the packages currently installed. In some cases, a service needs to be stopped. (For example, to upgrade **sendmail**, you would need to stop it, replace the program and then restart it.) This is all done automatically.

Debian deviates from the common RPM packaging format (although it can install RPMs) by using its own .deb format. The .deb format is the most versatile

and includes dependency checking as well as pre- and post-install and remove scripts. This is why the sendmail example in the previous paragraph can be handled automatically.

The most difficult thing about Debian is the initial installation. Or, put another way, fear of **dselect**, the installer program. The design of dselect is old, and while it made sense when there were only 50-100 packages in a Linux install, it is out of control now that there are around 1000. A replacement for dselect is being developed and will be available in Debian 2.1.

Versions of Debian (with limited applications/utilities) are available for Digital Alpha and M68k.

S.u.S.E.

S.u.S.E. is a German distribution with an installation "look and feel" similar to Caldera. It also uses the RPM package format and offers a save/restore configuration option during installation.

Two things make S.u.S.E. stand out from the others. First, XFree86 support tends to be better than other distributions because S.u.S.E. works closely with the XFree86 team. Second, there are more applications and utility programs in this distribution. A full installation takes over 2GB of disk space.

YAST, the install/administration tool, can handle .deb and .tgz packages as well as RPMs. Also, upgrades are quite easy and can be performed by putting in a new CD or pointing YAST at the files and telling it to perform the upgrade.

Which Do I Choose?

It depends. I have one system running Caldera, three running Red Hat (a PC, a Digital Alpha and a SunSPARC), two running Slackware, one running S.u.S.E. (a laptop) and quite a few running Debian. (Yes, I personally own too many computers.)

Further, there are problems with all the distributions—not the same problems, but problems nevertheless. As a result, I don't see a perfect answer—yet. This is not to say they don't work—just that each has its inconsistencies and limitations. They all suffer from the lack of a common administration tool.

At USENIX in 1997, Caldera announced a project called COAS (Caldera Open Administration System). The discussion at the conference showed there were more concepts to consider and a lot of implementation work before COAS could offer a uniform installation system that would meet the needs of the majority of Linux users.

Today, for a general-purpose system I tend to install Debian. I do, however, install other systems for other purposes. For example, I have S.u.S.E. on a new laptop because the volume of software included makes a more impressive demo system.

A better question is, “which one should you choose?” The answer is still, “it depends.” Here are some hints to help you along the way:

- If everyone you know is running a particular distribution and you are a newcomer, use the same one they do.
- If you like to roll your own—that is, you expect to compile and install everything yourself—Slackware is probably for you.
- If you want to “go with the crowd” today, install Red Hat.
- If you want “everything”, install S.u.S.E.
- If you need the most “commercial” looking product or you are a VAR (value-added reseller), pick Caldera.
- If the politics of free software is important to you and/or you want to get involved in development of a distribution, pick Debian.
- If you have a bunch of systems you need to interconnect and upgrade, pick Debian or hope Caldera gets COAS completed.

Conclusion

There is my input. Ask any other Linux user, and you will probably get a different opinion from mine. If you are not sure you have the right answer, there are some things you can do to make it possible to change distributions in the future with minimal impact.

- Make /home a separate file system. Then, if you change distributions, you don't have to save and restore your files. This also means you could have multiple distributions on one computer and share /home between them.
- Select hardware supported by most distributions.
- If you need to add applications that don't come with the Linux distribution, try to get ones that come with source code so you can upgrade them and port them to different distributions.
- Start with a Linux archive CD set (such as InfoMagic's Developer's Resource). That will give you at least three distributions (Slackware, Debian and Red Hat) with which to play.

Good luck and happy Linuxing.

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

Archive Index Issue Table of Contents

Advanced search

LINUX DISTRIBUTIONS COMPARED

LJ Staff

Issue #52, August 1998

Nine distributions in one easy chart.

July 1997

	Caldera OpenLinux Base 1.2	**Caldera OpenLinux Standard**	**Debian**	**LINUXW 2.**
Price	$59.00	$199.00	$0.00	$9.
Target Consumers	CDEHOV	CDEOV	CDEHOV	DE
Kernel	2.0.33	2.0.33	2.1.0	2.0.
Binary Format				
Primary/Other	ELF	ELF	i386,68k,Alpha,Sparc	X
Hardware Required				

Minimum Install	386,8,40	386,8,40	386,4,40	586,6
Minimum Use	386,8,40	386,8,40	disk20	486,8
Recommended Use	486,16,300	486,16,300	486,8,850	586,16
Packaging				
yes	yes	yes	yes	ye
Interdependencies	no	no	yes	no
Package Format	RPM	RPM	deb,RPM,.tgz	RPM .tg
Source Code Packages	yes	yes	yes	ye
Management Interface	X Windows or command line	X Windows or command line	character terminal	ye
Group/Subgroup Packages	no	no	yes	ye
Boot Media				
Boot Floppy included	yes	yes	no	no
Boot images to select from	module-based no limit	module-based, no limit	6	5

Floppies required	none	none	0	0
Run from				
CD-ROM only	no	no	no	no
CD-ROM mostly	yes	yes	no	ye
Configuration				
Custom X	yes	yes	no	ye
Network	yes	yes	TCP/IP,SMB,NFS,Appletalk,AX.25	ye
SLIP/PPP	yes	yes	yes	ye
User/Group	yes	yes	group-per-user	ye
Filesystem	yes	yes	e2fs standard	ye
Printer	yes	yes	yes	ye
Runlevel/init	yes	yes	sysVinit	ye
Install				
PCMCIA Ethernet	yes	yes	yes	ye
PCMCIA CD-ROM	yes	yes	yes	ye

from CD-ROM	yes	yes	yes	ye
from Floppy	no	no	yes	ye
from FTP	yes	yes	yes	ye
from local filesystem	yes	yes	yes	ye
from NFS	yes	yes	yes	ye
from Tape	yes	no	no	no
UMSDOS from Linux	no	no	no	ye
UMSDOS from DOS	no	no	no	ye
Media available	yes	CD-ROM	yes	CD-R
Documentation				
Custom manual included	yes	yes	yes (on CD)	ye
3rd party books/ manuals	no	yes	yes http:// www.linuxpress.com	ye
Extras				

Non-GPLed extras included	BRU Backup, StarOffice, Netscape Communicator, Novell Netware client, LISA	BRU Backup, StarOffice 4.0, Communicator, NetWare client, FastTrack Server, AG Adabas	none	FreeB NetB
Support				
Included	5 incident e-mail support	5 incident phone and email	lifetime free e-mail support	
Optional	support contracts	Support contracts available	paid commercial support	ye
Special certifications	forthcoming Caldera training	Forthcoming Caldera training	POSIX-compliant	nc
Sales method used	DFX	DFX	DF	X

LEGEND

Target Consumers:
C Commercial user
D Developer/Engineer
E Desktop/End user
H Hobbyist/Enthusiast
O OEM
V Value Added Reseller

Sales Method Used:
D Distributor
F Free via FTP
X Direct

Hardware Required:
Values are CPU, RAM (RAM for X), Disk

Archive Index Issue Table of Contents

Advanced search

Encapsulating IP Using SCSI

Ben Elliston

Issue #52, August 1998

Mr. Elliston is working on a protocol for using SCSI devices to network Linux clusters in order to transfer data at high speeds.

I was introduced to the UNIX operating system about seven years ago, and I soon became familiar with the networking companion to UNIX: TCP/IP (transmission control protocol/Internet protocol). As time progressed, I evolved from being a user of the UNIX command-line TCP/IP utilities, i.e., TELNET and FTP, to gaining an understanding of the internal workings of the protocols.

One point, reinforced by every book on TCP/IP I have read, is that IP was designed to be encapsulated in almost any of the available data link protocols. This design makes it an inter-networking protocol; it is inconsequential that a computer on the opposite side of the globe is connected to a token ring and your machine is connected to an Ethernet. I found this concept so impressive that I examined the various types of existing IP encapsulation. At the time, there was IP in IP, IP in IPX, IP in PPP, IP over Ethernet using NCR's WaveLAN spread-spectrum network adapters and others.

Before ATM (asynchronous transfer mode) and 100 Mbps Ethernet were readily available, I started thinking about what other bus networks existed for networking computers. There was ARCnet and token ring, but these media offered throughput capacities comparable to Ethernet. Moreover, I was interested in experimenting with coarse-grained parallel processing using a set of cheap PCs sitting in the same room where applications not only had to dispatch jobs, but also exchange large data sets in order to accomplish their tasks. In this situation, network latency was not a great concern, but throughput was.

Perhaps due to its analogous operation to Ethernet, the SCSI (small computer system interface) protocol popped into my head. It was very fast—SCSI-2 adapters were commonplace at the time. SCSI shares some attributes of

Ethernet, making it suitable as a network data link: each “station” has an identifier, only one “station” can use the bus at any time, and each end of the bus must be terminated with a terminator of a characteristic impedance. SCSI provided a miniature Ethernet, only much faster.

I acquired the ANSI SCSI standards documentation and started doing the background research that would be necessary to undertake such a project. After a great deal of reading and advancing from SCSI-1 to SCSI-2, I started thinking about a design which could elegantly handle the encapsulation of IP version 4. I immediately recognized the forthcoming issues of IP version 6, but chose to ignore them, given that I wanted to get something running immediately. I also reasoned that I wouldn't be doing much harm by developing yet another protocol that would be disrupted by IP version 6.

My design led to the RFC (Request for Comments) draft document entitled “IP Encapsulation over the Small Computer Systems Interface”, which can be found at ftp://ftp.internic.net/rfc/rfc2143.txt.

Background of SCSI

SCSI is a peripheral interconnection technology designed to offer hardware manufacturers a standardized protocol and hardware description in order to build peripherals and computers which can be interconnected. For example, the Apple Macintosh was an early mass-produced computer which allowed the connection of SCSI devices.

SCSI devices communicate with each other by sending data packets across a shared bus. The device uses a hardware handshake to acquire the bus—all other devices must be silent while another device uses the bus. Unlike Ethernet, the bus is not accessed using a collision detection mechanism. Instead, devices follow a stateful algorithm to acquire the bus. When idle, the bus is in a state, or phase, known as the *bus-free phase*. If a device wishes to access the bus, it enters an *arbitration phase*, but only if the bus was previously in the *bus-free phase*. Clearly, there exists the classic problem in mutual exclusion where two devices check the state of the bus, both finding it in the bus-free phase, and go into arbitration. In this situation, the device with the highest SCSI ID always wins. This could prove significant when designing a network of machines running IP over SCSI.

After arbitration, the device enters a *selection phase* in which the target SCSI device's ID is placed on the data bus. The *command phase* is used to transfer the command data to the target. The *reselection* phase is entered when the target device wishes to respond to the initiator. This allows the bus to be used by other devices while a device is performing its task. The *data-in* and *data-out*

phases are used to actually transfer data between the initiator and the target. The *message-in* and *message-out* phases are available to transfer additional control information between the initiator and the target. The *status* phase is used to transfer a status byte from the target back to the initiator to indicate the result of the operation. For instance, a tape drive might return a status code to indicate that the media was not loaded.

Design

At the beginning of the project, I specified some overall goals. These goals have had a major impact on the scope of my “IP over SCSI” project. Some people found items worthy of criticism—and on occasion, they were right. The main thing is to realize that some of the issues raised just didn't fit the scope of the current project. They will be addressed in a later implementation. The goals I set were:

- Take a purist's approach and develop a means of carrying IP datagrams across a SCSI bus. This means that the limitations of SCSI such as the number of addressable stations would have to be accepted and that larger networks would need to be constructed using conventional strategies such as situating IP gateways between these small SCSI networks. This has an interesting consequence which will be discussed later.
- Develop a protocol that is simple to specify and easy to implement.
- Implement the protocol within the Linux kernel as a modular network interface (in the sense that it can be loaded and unloaded using the kernel module tools). My reasons for using Linux are fairly obvious: PC-based SCSI adapters are much more readily available than SCSI adapters for any other system, and the Linux kernel source code is freely available to study and modify. Furthermore, most of the kernel developers are happy to correspond via e-mail to explain chunks of source code or areas where documentation is lacking.
- Implement the network interface in such a way that it would operate correctly regardless of the model of a SCSI host adapter. This may introduce further performance penalties, but is obviously desirable for most applications.

Implementation

Given these design goals, I developed a network driver which had the following attributes:

- The Linux SCSI mid-layer was utilized to satisfy the requirement of interfacing to host adapters regardless of manufacturer. This undoubtedly

raises some performance issues, but I have not yet identified them. The SCSI mid-layer does not acknowledge the requirement to initialize a host adapter into *target mode*, which means, unfortunately, that every Linux low-level SCSI device driver will require modification if it is indeed to be capable of ever supporting target mode or supporting IP over SCSI. I have modified the Adaptec 1522 device driver in this way, but there appears to be a lot of work involved in modifying all of the Linux SCSI drivers.

- IP datagrams can be transmitted from any SCSI host adapter to any other host adapter. These adapters may potentially be in the same host or, more likely, in independent hosts.

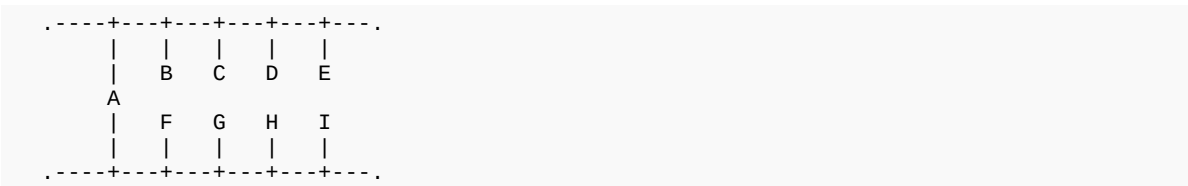
Applications

When I designed IP over SCSI, my intentions were to permit a number of closely situated machines running Linux to communicate using their existing base of software applications without modification, but at much higher speeds. This has minimal value, however, as networks such as Ethernet seem to serve most people's needs.

Other applications, which have not yet been fully exploited, could benefit a great deal from high-speed interconnectivity between hosts. I was recently a witness to a demonstration of the PVM (parallel virtual machine) manager running a massive computation on 31 Pentium-based Linux machines, and we observed that the bottleneck was the network used to transmit units of “work” and the subsequent results between the machines.

I, therefore, see that IP over SCSI has a number of immediate applications:

- A high-speed networking facility for general-purpose applications (e.g., using it as a dedicated network for file sharing with NFS in a research laboratory), while Ethernet could be used for all other applications.
- A means of connecting existing, closely situated machines for high-speed applications such as FTP mirroring or Web search engines.
- Clustering and coarse-grained hypercubes constructed using the inter-networking concept: small networks of hosts connected via SCSI and interconnected to all other such networks via one or more SCSI interconnections. Here, each multi-homed host, equipped with multiple SCSI adapters, acts as an IP gateway between the connected networks. Structures such as these are conceivable:



Here, hosts [B-E] can communicate with hosts [F-I], despite the fact that a SCSI-1 bus, for example, is unable to support a total of nine hosts.

Getting more creative:

```
A---B---C---D---E---.
|
F---G---H---I---J---.
|
K---L---M---N---O---.
|
P---Q---R---S---T---.
|
U---V---W---X---Y---.
```

This arrangement can naturally be extended to three dimensions by, at the bare minimum, adding a third SCSI interface to the gateway hosts {A,F,K,P,U}.

Future Directions

The encapsulation protocol for IP over SCSI has been documented and drafted a number of times and has passed through the Internet Engineering Task Force and is now published as a RFC document (RFC 2143).

There has been a good deal of interest in this concept. Another Linux user and recent computer science graduate, Randy Scott, has implemented the IP over SCSI protocol with success. His project does not exactly meet the protocol given in the RFC, but it does prove that the concept works. Randy's work, however, illustrates that there is an issue of performance when it comes to IP networking in the Linux kernel, most of which was beyond his control. It is understood that there is some doubt as to whether a network interface could have a maximum transmission unit (MTU) of 64KB.

My own implementation has not been getting as much attention from me as I would like. Until recently, work was progressing well. I have a modular network interface which can be brought on-line using **insmod** and **ifconfig**, and IP packets can be sent onto the SCSI bus and the correct SCSI ID selected using my implementation of an address resolution protocol (ARP).

The next step is to verify the modifications made to the device driver for initializing target mode, then receive data from the SCSI bus and pass it up the protocol stack. I would be grateful to receive any help in completing this project from interested individuals.

Ben Elliston is a software engineer currently working for Cygnus Solutions. His interest in computers just gets him into trouble, so in his spare time, he enjoys rock climbing, mountain biking, playing the guitar and spectating at rallies. He can be reached at bje@cygnus.com.

A First Look at KDE Programming

David Sweet

Issue #52, August 1998

Mr. Sweet teaches us how to write an application for the KDE desktop—for the experienced GUI programmer.

The stated goal of the K Desktop Environment, KDE, is to provide a free, user-friendly desktop for Linux/UNIX systems. The project's participants began by providing a window manager (kwm/kpanel) and a file manager (kfm) and retrofitting popular X applications (e.g., Ghostview, xcalc, ezppp), giving them a common look and feel based on the Qt widget set from Troll Tech. As time has passed, the KDE libraries (which provide UI elements and file management services) have grown in functionality so that developing KDE compliant applications is not only simple, but enticing.

The program presented in this article, **khello**, should provide you with a good foundation for writing your own KDE application, or providing a K-UI for your favorite UNIX application. To compile the program, you need to install Qt 1.3 and KDE. For details, see http://www.kde.org/beta1.html. Although KDE has just been released in only beta form, it is quite stable and usable. It has, in fact, been the primary desktop on my home computer since well before the beta release (alpha releases and frequent development “snapshots” have been available for some time).

Like so many before me, I have chosen to present a program which says, “Hello World!” to its user. Nevertheless, the basic elements of a KDE application are present: a menu bar, a toolbar, and an “about” box, as well as the functionality provided by KApplication and KTopLevelWidget. The UI components are provided by classes in libkdeui; KApplication belongs to libkdecore.

KApplication is the base class from which KDE applications are derived. It handles simple session management, access to the application icon, the help file, and configuration and locale information. The KApplication constructor takes **argc** and **argv** as arguments and strips away anything used by X11 or by KApplication and updates **argc** so that you can go on to process your program's options. The macro kapp is defined as **KApplication::getApplication()**. This returns a pointer to the KApplication object. You can use **kapp** anywhere within your program after you have created a KApplication. So, of course, you can only create one KApplication object. The GUI is started with **app-exec()** and returns when **app->quit()** is called. Notice that your instance of KApplication need not be told about your KTopLevelWidget. In our case, this means KHelloTW which is derived from KTopLevelWidget. There can be only one KTopLevelWidget, and it informs KApplication of itself.

The basis of the user interface is provided by KTopLevelWidget. It will manage a menu bar, multiple toolbars and a status bar, you don't need to resize or position these objects; just create them and fill with the appropriate information. (Some session management is also handled by KTopLevelWidget, but this topic won't be discussed in detail here.) You should derive a class from KTopLevelWidget, as KHelloTW is, and set up the UI (in the constructor, for example) before calling **show()**. This way, your window will not appear and then “flash” as its child widgets are created and/or rearranged. You must call show() for your KTopLevelWidget before calling app->exec(), since it isn't shown automatically.

Let's look at khellotw.cpp (Listing 1) to get an idea of how to work within KDE and Qt. First we create the file menu. We've declared “file” as a pointer to an object of type QPopupMenu and created the object with “new” so that the file menu will be available for the life of KHelloTW (which is, in this case, also the life of the program). The method **insertItem** places an item on the menu. In this case, we have only “Quit”. The ampersand in the argument **&Quit** tells Qt to underline the letter **Q** and, when this menu is visible, to allow the user to select “Quit” by pressing **Q**. The last argument **kkeys->quit()** says the user can press the key combination defined in **KStdAccel::quit()** (**ctrl-Q**) at *any* time to exit the program (this is called an “accelerator key”). The class KStdAccel contains various standard KDE accelerator keys. We use these definitions when creating menus to be consistent with the KDE look and feel. The other two arguments tell Qt which method (**slotQuit()**) to call when “Quit” is chosen and within which instance of the class to call that method. This system for getting messages from the GUI is called “Signals and Slots”. If we use the Qt precompiler (called **moc**

for Meta-Object Compiler), we can use these signals and slots as if they were a natural part of C++.

Using moc, we get three new C++ keywords: **signal**, **slot** and **emit**. In khellowtw.h, several methods are declared under the heading “public slots:”. These can be called directly, like normal methods, but they can also be connected to signals. Signals are emitted whenever any GUI event occurs. All the slots which were connected to that signal are called at this point, and may even be called with arguments so that information can be passed between classes. (For details on how to use signals and slots, and for a definition of emit, see the Qt documentation.)

If we look at the **toolbar->insertButton** line in khellotw.cpp, which places a QPushButton on the toolbar, we see

```
SIGNAL ((released()), this, SLOT (slotHello()))
```

This tells KToolBar to connect the signal **released()**, which is emitted by QPushButton when the button is released after being pressed down, to the slot **slotHello()**, which puts the phrase “Hello world!” up in our window. Note that we chose the QPushButton signal **released()** over **pressed()** because the user expects the action to occur when he releases the button, not while holding it down. You could try changing this to see the difference in the feel of the interface.

The moc precompiler generates some C++ code from your header file and creates a .moc file. This .moc file should be included once and only once in one of the files that implements that class. Notice that khellotw.cpp includes khellotw.moc but main.cpp (Listing 2) includes khellotw.h (Listing 3).

To give KHello the look of a KDE application, we have added a KToolBar and used one of the pixmaps supplied by KDE for the button. To construct the path to the button pixmap, we've used **kapp->kdedir()** rather than hard coding the path, because users may install KDE in different places in their file systems. For the rest of the path, refer to the KDE File System Standard (or KFSSTD at http://www.kde.org/fsstnd.html). There is also a help menu on our menu bar. KApplication provides you with a base help menu in **kapp->getHelpMenu()**. This menu includes “About KDE...” and, optionally, “About Qt...” entries which tell the user about the underlying software, and a “Help” entry which will start the KDE help viewer (**kdehelp**) with the argument

```
kdedir()+ "/share/doc/default/khello"
```

which refers to the default KDE help file/directory. If you'd like to see this in action, simply create the directory, place an HTML file called khello.html in it,

and choose “Help” from the khello Help menu. When invoking help this way, KApplication uses the string passed to its constructor as your application's name (in this case, **khello**). So there is no need to tell KDE where your help file is; it is determined by the KFSSTD and is /share/doc/default/appname/appname.html.

Listing 4. Makefile

Similarly, you can give your application an icon (which will appear in the task bar next to your application) by placing an appname.xpm file in /share/icons/. We've added a separator and “About KHello...” to the bottom of the menu. Your “about” box should display at least the title, author and contact information (for bug reports, feature requests and general praise of your work). KHello's simple “about” box is a KMsgBox. This is a modal dialog box, meaning that it is the only window of your program which will respond to user input during the time it is shown. When the user selects “OK” the box disappears, the function call returns and the program continues. It is not necessary to create an instance of KMsgBox for simple dialogs. The method **KMsgBox::message()** is static and can be called directly as is done in **KHelloTW::slotAbout()**.

Application communication via drag-and-drop is an integral part of a cohesive desktop. If your application works with data files, it should accept URLs dragged from kfm (the KDE file manager) and process them accordingly. This is simple to do and khello demonstrates this by displaying any URL that is dropped on it. To accept URL drop events, declare a widget a “drop zone” by creating a KDNDDropZone object and connecting its **dropAction()** signal to a slot which will process the event. The KDNDDropZone method called **getURLList()** will return a **QStrList** (a Qt utility class which manages a list of strings) containing one or more URLs which were dropped in a single drop event. The function **slotDropped()** sets the text of “label” to the first URL in the list.

There are several ways to shut down an application. The user could choose “Close” from the system menu (a right-click on the title bar brings this up) or click on the close button (an “X” on the title bar), or one's application may be closed by the window manager when the X session is terminated. All of these call the **closeEvent()** member function of your KTopLevelWidget. So this is where you should ask the all-important question, “Would you like to save changes to _____?” Then call **kapp->quit()**, which tells your KApplication you are done and the **app->exec()** call in **main()** returns. To keep your program organized, you should force a closeEvent() call if you offer the user an alternative way of exiting your program (like choosing “Exit” from the “File” menu). This way, all pre-termination code is in that one place (closeEvent()). You can force a closeEvent() by calling **close()**, which is a member of KTopWidget. (In

fact, it is a member of QWidget from which KTopWidget is derived.) This is done with KHelloTW::slotExit().

I encourage you to experiment with KHello and alter the code to learn about other KDE classes. A good place to start is **KConfig**. This class allows you to read and write from an application-specific configuration file stored in ~/.kde/config/. You should save the default program options in closeEvent(), then reread and set them upon construction of your KTopWidget.

KLocale is another important class for KDE applications. It helps you to translate your application into other languages by reading string literals (like “File”, “Help” or “Hello world!”) in from a file. KLocale will choose the appropriate strings based on the user's locale settings. You need only provide the text. Since KDE is developed and used by people all over the world, it is a good idea to translate your application; the widespread interest makes it easier to find someone to help with the translation.

For further information on programming for KDE, I recommend going through the Qt tutorial at http://www.troll.no/qt/tutorial.html and visiting the KDE developer's center at http://www.ph.unimelb.edu.au/~ssk/kde/devel/. Here you'll find helpful tips on KDE programming as well as documentation for the KDE libraries and ideas for new KDE applications. It is in the spirit of the KDE project to announce your intention to create an application on the KDE mailing list so that effort is not duplicated. You'll also find out how interested others are in using your proposed application and, most likely, receive lots of suggestions for program features. Good luck with your KDE programming.

David Sweet is a third-year physics graduate student at the University of Maryland and a participating programmer in the KDE project. He is currently working on an interface to Ispell and a plotting package called KPlot (http://www.glue.umd.edu/~dsweet/KDE). He can be reached via e-mail at dsweet@chicago.umd.edu.

Archive Index Issue Table of Contents

Advanced search

Linux Stampede

David Haraburda

Issue #52, August 1998

The new kid on the block—Stampede Linux.

Stampede Linux was created because of our dissatisfaction with other distributions. We found that while other distributions had many useful features, none of them had everything we wanted. While the development team recognizes perfection is an unachievable goal, the driving force behind the development of Stampede Linux is to create a perfect distribution.

The Stampede Linux Project was founded by Matt Wood, who heads the development team and is the backbone of everything relating to Stampede. His original intention for Stampede was for it to be a Pentium-compiled rebuild of Slackware—a private distribution for use by himself and a couple of friends. However, Stampede has grown to be much more; it dropped its Slackware ties early in the development process.

Perhaps the most frequent question we're asked during this development period is: “Why another distribution?” Several reasons provide the answer:

- None of the current distributions are optimized for newer machines. We don't believe Pentium/Pentium Pro/Pentium II/K6 machines should be running code written and optimized for 386 or 486 machines. All of Stampede's binaries and packages are compiled using the Pentium GCC Compiler.

- There isn't a distribution centered mainly around up-to-date and current packages, as well as security. We believe that having a secure system (in conjunction with updated packages) is the only way a distribution should be created. Rather than giving the user the option to install packages to increase security, Stampede will be as secure as possible, and give the user the option to install packages to decrease security.
- Although most distributions are fairly easy to install, we find that for the new user they aren't easy enough. If the Linux revolution is going to take place, simpler and more intuitive installation and configuration programs must be developed.

These reasons have led to the implementation of several key features. First off, all packages are compiled with the Pentium GCC Compiler, a variation of the GCC Compiler with optimizations for Pentium class chips. This is important because PGCC increases performance of a Pentium-compatible system and usually results in a 10-30% faster system. The Pentium GCC compiler is also fully compatible with i386+ processors and clones. (Information on PGCC can be found at http://www.goof.com/pcg/).

All Stampede stable binaries are also compiled **without** debugging (**-g**) information, delivering a smaller, faster binary. We are considering the release of a separate package which includes debugging information, for those who want it.

Stampede Linux also uses gnulibc2 (libc6), a much faster version of the standard C libraries. This version includes a more stable interface, thread-safe functions, an improved math library, new functions from POSIX and XPG4.2, and is 64-bit clean. However, it is still under development and is available only for GNU-based (both Hurd and Linux) systems. More information on GNU libc can be found at http://www.gnu.org/software/libc/libc.html.

The Stampede Linux Package (SLP) Management system is also progressing quite nicely. Unlike some other distributions, using this package is totally optional. If you wish, you can stick with gzipped tar (or bz2) files and not use the SLP files. However, we believe that most users will want to use the package format because of its portability. The development team plans to have automatic upgrading of packages implemented by version 1.0 of Stampede.

Stampede plans on using BSD-like init scripts to configure all of the system run-level maintenance, but developing these scripts does not yet have a high priority. Even though these scripts are easy to configure, we still plan to create a set of configuration tools which allow a simple, straightforward way of setting up and changing the init scripts.

Another feature worth mentioning is the boot/root disks. With Slackware, finding and downloading the correct boot disk seemed to annoy almost everyone on the development team. So, we have decided to create a third disk, a modules disk. Anything that can be compiled as a module instead of being built into the kernel will go onto the module disk. This will include CD-ROM, Ethernet and hard disk drivers. Doing this should make it easier for newcomers to Linux to find the right boot disk.

The setup program is also notable. We have designed it in such a manner that it auto-detects the packages on the source media, and then asks you which ones to install. This method makes it much easier for us to add and update packages and package sets, as well as for the user to create or install their own. There will also be an option to select from a number of different automated installs. This will be a big plus for those who are new to Linux, and also to those who prefer not to answer questions during the install.

The development process of Stampede is quite unique. The Stampede Linux development team consists of people in different areas of the world who collaborate their efforts over the Internet by e-mail and IRC. While it does happen that tasks are assigned to certain individuals, most of the time one of us comes up with an idea, or gets one from a user or mailing list, and then works on it. If it gets the official okay from Matt, it goes into the distribution. Personal preference is a factor in determining which tools and programs we choose to include in the distribution, but we try to create and include programs which are easy for the new user, yet powerful enough for the more advanced.

During this development stage, we try to maintain a good balance between functionality and stability. We feel that as long as a program isn't too buggy or awkward to the user, it is worth including. We don't stick with software just because it's in other distributions. We realize that by using programs such as gnulibc2 and PGCC, and our new package format, we are taking a potential risk that users who have become accustomed to the older software may run into trouble at this time. However, we believe the advantages greatly outweigh the disadvantages. We also have confidence in our method because of our plans for SLP. With the automatic package updating mentioned earlier, there should be no problems.

Like other computer programming projects, we have obstacles we are trying to overcome. One of the main obstacles we face in our development is that all testing must be done on our own personal computers. Since at this point we have no money to buy equipment, it is hard for us to test Stampede on different computers. We need testers with different equipment. However, we still plan to become a non-profit organization, dependent on user efforts and donations to help us further development.

Odd bugs seem to be giving us a lot of trouble at this time. Since we can't always tell whether something is going wrong because we placed it in the incorrect place, compiled it wrong or coded it badly, it is hard for us to find bugs in the actual software. Thankfully, this seems to be less and less of a problem as development progresses and the system becomes more stable as a whole.

Any and all ideas given to us are taken into consideration. If it is worthwhile to the user to have in the distribution, it is worthwhile for us to include. It's hard to say what we have planned for future versions, because any new idea we have, we try to squeeze into version 1.0.

However, here are some things we hope to see in Stampede's future:

- Ports to different architectures: we recently found someone to help port to SPARC, and we also have members of the development team who plan to start on a Power PC port.
- A central system configuration utility: this will most likely be for the X Window System, written in GTK (which all our X utilities use). It will include sections to add/remove users and groups, configure the init scripts and much more.
- A more automated package update process: as mentioned earlier, automation is something we strive for, as we believe this will not only help new users, but will also give experienced users less worry. We are beginning development of the Stampede Linux Package Daemon (**slpd**), which will optionally query a central server at given periods of time and check for package updates, then download and install them.

Development of Stampede Linux is moving ahead at an incredibly fast pace. We have a very talented team of developers, and all are devoted to making Stampede Linux the best distribution available. The creation of Stampede is only 50% coding, because putting it all together and making sure it all runs smoothly has proven to be an equally big task. However, we take things one day at a time and deal with each obstacle as it comes. We aren't in this for the money or to outdo anyone. We are fully supportive of the Open Source cause. We simply want what you want—a distribution that is fast, stable, secure and robust. It is our goal to prove that there is always room for improvement.

David Haraburda (gerwain@beer.stampede.org) is a Linux fanatic who currently resides in Fort Worth, Texas. Information on him and the other developers can be found at http://www.stampede.org/people.html

Archive Index Issue Table of Contents

Advanced search

Muscle Flexes Smart Cards into Linux

David Corcoran

Issue #52, August 1998

The newest kind of card for your pocketbook offers better security for the information it holds.

Credit card companies successfully marketed the silver card, the gold card and the platinum card. Precious metals represent wealth, and we were supposed to associate that notion with something less tangible—card security.

In today's society, better security for electronic commerce is a priority for corporations, banks and merchants. Even as computers become faster, methods of authentication once thought secure become easier to subvert. At the same time, modern communications allow hackers to advertise their exploits, making details of such activity easy to come by, even for amateurs. An easy-to-use exploit script is usually just a web site away. As current security methods become easier to crack and subvert, developers search for different and better ways of authentication.

Survival of the Weakest

We all use magnetic-stripe cards every day to get money from the bank, to shop, to fill the gas tank and to enter the office building. It's not uncommon for a person to carry ten or more different cards at any given time. Are the cards secure? Consider the following common credit card scenarios:

- You pay your restaurant bill with a credit card and leave the receipt on the table.
- You make a department store purchase with a credit card and later throw the receipt in the trash.
- You send your mother flowers purchased over the Internet using a credit card number.

Card receipts contain all the necessary information for anyone to assume the same purchasing power (indeed, the same identity) as the actual card holder. In any of the above scenarios, it is only a matter of time before your identity and potentially your bank account is compromised. Authentication using magnetic-stripe tokens is plainly insecure.

Another example of weak authentication is the computer password—the most common method of authentication in computing today. Being the only person who knows a secret word or pass phrase gives us a misplaced sense of security about data access. We don't think about the adolescent cracking our password while we sleep, yet dictionary-based password crackers (software that tries common dictionary words) are available on the Internet. A software package such as Crack 5.0 easily reveals passwords in a matter of minutes—at most a few hours—that were once thought secure. Most hackers obtain initial access into a system through accounts with no password at all or a password set to match the user name. (Many system administrator packages set the default account name to match the user name, and the default never gets updated.)

You can strengthen current security mechanisms, but the ultimate solution may be an entirely new design. Soon magnetic-stripe credit cards, passwords and PINs could be things of the past, replaced by token-based authentication systems enabled with biometric sensors—a secure token storing cryptographic certificates and keys, and terminals with biometric sensors to qualify key access (e.g., from your fingerprint).

The Smart Alternative

One such token is the smart card. Smart card technology has been reliably demonstrated in Europe for many years. A smart card is a special-purpose storage device about the size and thickness of a credit card, containing a thin microprocessor capable of storing data. When inserted into a suitable reader, the microprocessor is powered up, and instructions are passed between the host computer or terminal and the smart-card microprocessor.

Your wallet today: some cash, a few credit cards, an ATM card, a phone card, a driver's license, employee badge, a piece of paper with all your PIN numbers written on it and a reminder to buy milk.

Your wallet in the future: a single smart card with biometric authentication.

What can you do with that smart card? A single card has the potential to replace:

- bank cards, ATM cards, phone cards, gasoline cards, credit cards in general

- driver's license, passport, employment identification
- physical access to your home, your car, your office
- medical records and services

With biometrics, you might even be able to remember that you are allergic to dairy products or penicillin. Biometrics are measurements taken from a person to identify them later, such as fingerprints, retinal patterns, face photos, finger lengths, voice prints or typing and writing patterns. Measurements can be digitized for storage on a smart card and for use in conjunction with digital certificates and authentication.

Figure 1. Smart Card

Smart cards are true computing devices containing a CPU, ROM, EEPROM and RAM—all packed into a flexible plastic card. The technology of the microprocessor is comparable to that of a full-sized desktop computer in the late 1970s. Smart cards have an operating system, an I/O channel, static and dynamic memory and an instruction set used to program the card. In the future, the technology may evolve so that entire operating systems (such as Linux) could be run from the card. Right now, smart cards mainly provide secure, portable storage for cryptographic keys and a wide assortment of authentication information.

Smart Card Flavors

Smart cards, like ice cream, come in many different flavors, but there are two fundamental types: memory-only and processor-based.

Plain vanilla memory cards are typically used as electronic purses—for example, phone cards, vending machine cards or university campus cards that store cash value. The value can be decremented with each use and recharged at specialized machines; consequently, such cards are sometimes referred to as cash cards.

Processor cards enjoy a more varied deployment. They can contain a cryptographic coprocessor for authentication and file encryption. Some processor cards actually run Java binaries directly on the card using a built-in Java virtual machine (a subset of the one that runs on any full-sized computer)

to interpret the commands in the Java binary. These are referred to as Java cards. Most of a Java card's functionality can be exercised through Java classes, allowing the card to be programmed in almost any platform or hardware specification. Java cards also include some limited cryptographic functionality, which can be accessed through Java security classes. Java cards combine the ease of a familiar programming language with the robustness of the smart card.

Both memory and processor smart cards support two types of I/O: actual electrical contacts or radio-frequency induction.

The burgeoning need for secure transactions and e-mail privacy may be answered by the next generation of cryptography cards, such as the Schlumberger CryptoFlex smart card. Such cards are capable of a wide variety of cryptographic functions, including random-number generation, digital signing and encryption. These cards are typically used in an authentication process, where cryptographic keys associated with an X.509 certificate are stored on the card and unlocked for use with a PIN number. Or biometric measurements on the card are compared with those retrieved through a card reader equipped with biometric sensors. After authentication, a user can access the public/private key pair directly on the card and use it to sign or encrypt messages using the card's cryptographic coprocessor. This key pair never leaves the card. The cards can store more than one key pair and are capable of doing multiple cryptographic algorithms such as PGP, RSA and DES.

Unlike the magnetic-stripe card, which uses single-factor authentication (you have the card in your hand, therefore it must be yours), cryptographic smart cards actually run software right on the card. This allows bi-directional authentication between the smart card and the card-reader terminal, and when biometrics are involved, multi-factor authentication. One possible authentication scenario runs like this:

- The card reader terminal powers up the card and identifies it through the “Answer to Reset” function.
- The reader initiates authentication by transmitting a random number to the card along with a request for encryption.
- The card authenticates the user through a PIN or biometric measurement, and if successful, encrypts the random number using the private key stored on the card. The encrypted random number is returned to the reader along with the certificate of identity.
- The card-reader terminal obtains the public key (using a directory lookup or other database) and decrypts the random number using the public key.
- If the decrypted random number matches the one originally transmitted, the user and card are authenticated.

A similar process can even be used by the card to authenticate the reader terminal.

More about Card Readers

A card reader or terminal is required to power and communicate with the card. There are a wide variety of smart card readers on the market. They include contactless or RF readers, and the more common electrical-contact reader. These readers can be computers in their own right or can interface with any host computer via serial, parallel, USB and PCMCIA ports. Other reader-to-host interfaces are possible, even through a floppy drive. Some readers incorporate PIN pads or biometric sensors directly in the reader so that the PIN, pass phrase or biometric measurement never leaves the system.

Readers and terminals come in a variety of sizes, shapes, colors and functions. The Schlumberger Reflex 60 smart card reader is designed for PC applications —for example, secure business-to-business transactions between merchants and banks over the Internet. You could also have secure access to Web TV, games and many other applications.

Smart cards communicate with the reader through either the contact plates located on the plastic card or through radio frequency. In each case, the cards communicate through either the T=0 or T=1 protocol. T=0 is a byte-oriented protocol where an instruction byte is sent and an acknowledgment is received. This may be an error message or just an acknowledgment of the instruction. T=1 is a block-oriented protocol which sends a specified unit of data.

Like PCs, smart cards have a file system which contains a root or master file. All files are identified by two bytes. The master file on smart cards is identified by **3F 00**. Dedicated files are similar to sub-directories in that they allow files to have specific paths. Information is actually stored in what is called the elementary file. The elementary file can be defined in different ways, depending on how the user wants it subdivided for storage. The following diagram shows a simple smart card file system.

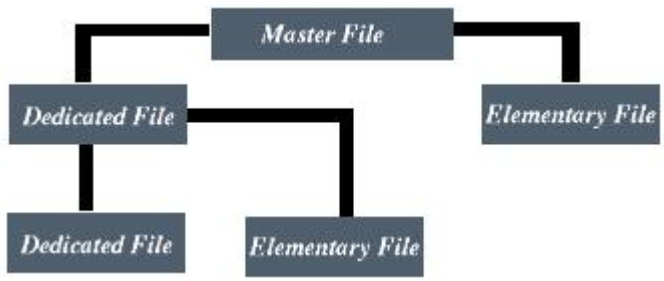


Figure 2. Smart Card File System

Smarter Cards, Bigger Muscles

As smart cards work their way into our lives through cellular phones, stored-value and authentication systems, they will also make their way into our general computing environment. Linux provides a secure multi-tasking environment perfect for smart cards. Project MUSCLE (Movement for the Use of Smart Cards in a Linux Environment) is a virtual team of developers working to integrate smart cards, readers and security in an open fashion for the Linux environment. As the acronym implies, the MUSCLE team is trying to integrate all necessary smart-card hooks into the Linux world. See sidebar for contact information.

Figure 3. Muscle Logo

Several different MUSCLE projects are underway, such as an effort to integrate smart-card tokens with Pluggable Authentication Modules (PAM), interfaces for scripting languages such as Perl, and creating drivers and implementing APIs for different cards and readers. MUSCLE expects the following benefits from these activities:

- Integrating smart cards into PAM will allow cross-platform authentication between an array of different operating systems.
- Perl integration provides another environment for smart card use, in particular for those accustomed to the ease of string manipulation in Perl.
- Standardizing card and reader APIs under Linux gives developers a common interface for integrating a wide variety of smart cards and readers with a wider variety of host computers.
- Creating a Linux cryptographic API will handle cryptography for other PKCS-11 type devices as well as for smart cards.

- Virtualization of the smart-card file system would allow a smart-card to be mounted as any other file system under Linux. This might allow the manipulation of the smart-card file system in an easy-to-use, familiar fashion.

Developing a means of using different biometric sensors with smart cards under Linux is another interest. Biometric fingerprint scanners such as American Biometric's hot new BioMouse and BioMouse Plus could easily make Linux the operating system of choice for high security applications such as banking.

MUSCLE posts all source code under the GPL on the web site http://www.linuxnet.com/smartcard/index.html, along with the author name, date and purpose. All code posted is covered under the GNU public license, allowing it to be freely distributed as long as the rules are followed. This is where open development occurs. For more information on the GNU public license, visit http://www.gnu.org/.

MUSCLE currently has support for the Schlumberger Reflex 62 and 64 card readers. These readers rely on the serial port for data communication and get their power from the keyboard or mouse port. The current drivers support all of the necessary functionality of the Reflex 62 and 64 readers, including PIN verification (Reflex 64 only). These drivers rely on the termios library for serial functionality and can be placed on any Com port. Currently, all Schlumberger CryptoFlex card functionality is supported, including ISO-7816 functionality and cryptographic functions (file reading and management, RSA signing, key and PIN verification, authentication, seek and other administrative functions).

Public interest in smart cards will spur technology growth and increase the need for a better user interface. An embedded operating system such as Linux can offer the ease of a UNIX-style operating system along with full functionality of the card. For example, to retrieve information about the file system directory structure, embedded Linux would allow a simple **ls** command to be used instead of the hex command **0xF0 0xA8 0x00 0x00 0x00**. Currently, smart cards are not capable of an embedded operating system. Until they can, familiar shell commands such as ls could be accomplished on the smart card by virtualizing the shell—interpreting shell commands into their smart card hexadecimal counterparts. This would provide an easy way to personalize the card to fit customer needs.

What will customer needs be in the next millennium? Some people predict less emphasis on materialism. Exercise machines might be as prolific as ATMs, and the gold card may be less attractive than one with more brains. Securing the future always involves flexibility and change. Smart cards appear to combine it all.

Resources

Dave Corcoran is an undergraduate at Purdue University, where he studies Computer Sciences, and is also a Knowledge Worker at Schlumberger Limited. His latest project includes the integration of smart card security tokens into the Linux operating system. He can be reached at corcoran@slb.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

XSuSE—Adding More to the XFree86 Offerings

Dirk H. Hohndel

Issue #52, August 1998

In mid-1997, S.u.S.E. started to release a small family of X servers, called XSuSE, based on XFree86 and freely available in binary form. This paper explains who is involved in doing this, why we are doing it, what exactly we are doing and what will happen next.

XFree86 is the freeware (or Open Source software) implementation of the X Window System for PCs running UNIX-like operating systems. XFree86 is largely independent of Linux. It was started as a freeware project in early 1992, shortly after X11R5 was released. At that time Linux was in its infancy, and the initial platforms supported in XFree86 were commercial UNIX implementations such as ISC SVR3 and Dell SVR4. However, by mid-1992, XFree86 was ported to Linux as well, and today the vast majority of XFree86 users are running Linux as their operating system.

Released in September 1991, X11R5 was the first official version of the X Window System that included support for PC UNIX systems. X386 had been donated to the X Consortium by Thomas Roell, who back then was working for SGCS, a consulting company that, among other things, offered commercial X servers for PC UNIX systems.

Unfortunately, the public fixes for X11R5 did not address the performance and stability problems which made X386-1.2, as included in X11R5, almost unusable for most owners of PCs running UNIX. The lack of patches to X386 brought a group of four people together with the intent to enhance X386. David Dawes, Glen Lai, Jim Tsillas and David Wexelblat started what they thought would be a small project for others to use. In July 1992, they released X386-1.2E.

Soon it became obvious that this project, creating a usable freeware implementation of X, would be much more than just an enhancement to X386, so the name of the project was changed to XFree86, indicating the (as we now

know, initial) platform that the software ran on and the team's central goal of providing a freely available implementation of the X Window System.

Since then, there have been about a dozen releases of XFree86. The latest is XFree86-3.3.2, released in March 1998. The number of supported cards is hard to keep track of; the number of cards and chip sets not supported by XFree86 is surprisingly small. XFree86 runs on most of the currently available UNIX-like operating systems on PCs. Besides Intel x86, Digital Alpha, Motorola Power PC and 68k are supported platforms with others in the pipeline. The number of people actively involved in the project is now close to 300.

This is quite a change from the small project started by the “Gang of Four”, as the initial founders of XFree86 were sometimes called.

In 1994, the X Consortium was working on X11R6 and the XFree86 team decided to participate in that development. In order to make that possible, a legal entity able to join the X Consortium was needed, and after discussing the alternatives, a non-profit corporation called The XFree86 Project, Inc. was founded.

This corporation is still maintaining the code and organizing the development of XFree86. I am a member of the XFree86 Core Team and Vice President of The XFree86 Project, Inc.

S.u.S.E. GmbH was founded in 1992, coincidentally the very same year XFree86 was started and Linux became more widely used. Since 1993, S.u.S.E. has been selling Linux distributions, first on floppy disks, later on CD-ROM. The latest release of S.u.S.E.'s Linux distribution, S.u.S.E. Linux 5.2, comes on four CDs with a detailed handbook and 60 days installation support. S.u.S.E. Linux is available in German and English versions; other national versions are under development.

Currently about 50 people are working at S.u.S.E., serving more than 100,000 customers worldwide. S.u.S.E. offers professional support, training, pre-installed and configured hardware, consulting and software development for Linux-based systems.

S.u.S.E. is committed to actively furthering freeware development and to supporting the Linux community.

Why

The XFree86 development model has always been a bit different from the philosophy used in large areas of Linux development. The main difference is that only a small group of developers (currently about 300) has direct access to

the most current sources. This development team creates rather well-tested releases of XFree86 which are then distributed, in source and binary form, to the general public.

There are many reasons for this setup. Among the more important is the support load that a system like XFree86 creates. Configuring XFree86 is difficult for many people, and setting up PC hardware can be quite challenging. Therefore, a significant number of users have questions or need help configuring XFree86. Making a rather small set of releases available to the public makes it possible to have a reasonably good idea as to which software is actually running on the system of someone who asks for support. Trying to support arbitrary development versions is simply not possible.

Additionally, there is a slight risk that development versions of the software may contain code that could damage the hardware. Today's monitors are far less sensitive to over clocking, and other components are more robust as well, so this argument is slowly losing its basis.

Nevertheless, the development model of XFree86 is still the same and not likely to change any time soon. It results in somewhat long intervals between new releases.

On the other hand, the generation cycle in the graphic-card industry is getting shorter and shorter, and the number of new cards and chip sets appearing on the market is growing rapidly. The shelf life of many cards is reduced to 6-9 months. This rapid turnaround in the graphic-card market means that many of the well-supported cards in XFree86 are no longer available, and many newly available cards are not yet supported in XFree86.

As a provider of a Linux distribution, S.u.S.E. is obviously interested in having available the best hardware support possible for its customers. In order to achieve that, there is a long-standing and very positive relationship between S.u.S.E. and The XFree86 Project. As part of this relationship, S.u.S.E. donated money as well as equipment to XFree86 in order to further the development of XFree86 and to make sure new versions of XFree86 can be released in a timely manner. Since 1995, S.u.S.E. has paid developers to work on XFree86 development, and in 1997 I worked almost full time on XFree86 for nine months while a S.u.S.E employee.

Even so, this help has only partially reduced the problems of XFree86 in keeping up with the new hardware appearing on the market.

What

In mid-1997, Elsa AG, one of the many graphic-card vendors actively supporting XFree86, suggested they could help develop a server for their new line of 3DLabs-based graphic cards. Since documentation for these cards was only available under a non-disclosure agreement (NDA), S.u.S.E. agreed to do the development and to donate the server back to XFree86 as soon as the NDA was lifted. (Unfortunately, this has not happened yet, which is the only reason this code was not integrated into XFree86-3.3.2.)

After a few months of development, S.u.S.E. made the first server for Elsa's cards, XSuSE_Elsa_GLoria, available. This marked the beginning of the small family of XSuSE servers. The server quickly became very popular, and over time was extended to support many non-Elsa cards and updated to support newer chip sets from 3DLabs. A major share of the development of this server was done by S.u.S.E. employees, but other XFree86 developers contributed significantly as well.

The motivation for doing more than just this one server came from the fact that Matrox released the documentation for their Millennium II card just weeks before XFree86-3.3.1 was finished. While time permitted a quick hack to get XF86_SVGA to work on the Millennium II, it was soon obvious there were some problems left unaddressed. Additionally, the AGP version of the Millennium II became available, which was not detected by the XFree86-3.3.1 server. S.u.S.E. decided to fix the most important problems in Millennium II support and add Millennium II AGP support. This was released as XSuSE_Matrox.

This started a flood of requests for servers for other, recently released hardware, for example, the Riva128 chip from NVidia, a newer version of the Mach64 series from ATI, or the AT3D and AT25 chip sets from Alliance Semiconductor. The XFree86 developers started to work on drivers for these chip sets, but a new release of XFree86 was months away.

Instead of telling people to wait, many developers wished to make their servers available. However, given the size of the XFree86 sources, releasing patches did not seem to be a good way to give the majority of the users access to these drivers. Additionally, this would have created a confusing mess of different versions available, something that XFree86 is trying to avoid. At this point, S.u.S.E. took over the coordination of releasing interim servers, helped to develop many of these drivers with its own employees and made the servers available to the public in the form of binary-only releases.

The XFree86 Project was willing to give permission for these releases under the following conditions:

- They were not to be called XFree86.
- They would be supported by S.u.S.E. and not create an additional support load for the XFree86 team.
- All code developed for these servers would be donated back to XFree86.

Since these conditions exactly matched the intentions of S.u.S.E., the creation and distribution of the XSuSE servers began.

What Next

Since then, many people have wondered if S.u.S.E. would begin to develop commercial X servers and become yet another player in that market (like Metro Link and Xi Graphics). This was never the intention behind developing these servers in the first place, and is still not among the options being considered by S.u.S.E. On the contrary, these servers are provided as freeware and can be freely distributed by anyone. S.u.S.E. explicitly encourages other Linux distributions to include these servers on their CDs. S.u.S.E. is supporting these servers regardless of whether the user has purchased S.u.S.E. Linux or another Linux distribution. Feedback from support is collected and provided to the developers. The source for these servers is part of the development source of XFree86.

As long as there is demand for early access to server binaries, S.u.S.E. will continue to make servers available in the XSuSE series. This is the case for XSuSE_Elsa_GLoria right now, and other servers will be added as new drivers are written before the next release of XFree86 is ready.

Additionally, S.u.S.E. is actively working on enhancing many other aspects of XFree86, most notably the configuration of the servers. Due to the size of this undertaking, we are hoping to do this as a joint project with XFree86, S.u.S.E., Red Hat and other companies distributing XFree86.

This article was first printed in the proceedings of Linux Expo 98.

Dirk Hohndel (hohndel@aib.com) is Vice President of The XFree86 Project, Inc. He was an employee at S.u.S.E. GmbH in 1997, and still works for S.u.S.E. on a freelance basis. His involvement with Linux started in November 1991, and he has been active in the freeware area ever since.

Archive Index Issue Table of Contents

Advanced search

UniForum '98 Report

Phil Hughes

Issue #52, August 1998

LJ's publisher flies to the east coast for the annual UniForum conference and spends more time at Linux track sessions than on the beach.

I am writing this article as I fly back from the UniForum Association Spring '98 Conference. This was only the second East Coast conference held by UniForum in its 17-year history. It was held in Ocean City, Maryland—a beautiful town by the Atlantic Ocean.

Ocean City, Maryland

UniForum is an advocacy organization that promotes the use of Open Computing solutions—or, to put it in succinct terms, the use of UNIX and UNIX-like solutions. UniForum and Usenix have tended to complement each other with UniForum being the conference for suits and Usenix the conference for T-shirts.

The conference was a two and a half day event that looked light on paper. Once there, I discovered there was too much to do in such a short time. For the first two days, there were three tracks: *New Open Software Development Model and Linux*, *Network Computing* and *Best of SCO Forum*. All tracks included

common Plenary and Keynote sessions. On the third day there were two three-hour sessions: one presented by the Open Group and the other by Linux International. I attended all the Linux track sessions as well as the common sessions.

Monday's Plenary session was Eric Raymond's presentation of his paper *The Cathedral and the Bazaar* which is credited with convincing Netscape to take their Open Source stance.

Eric Raymond

If you haven't heard the talk or read the paper, it is available on our Linux Resources page.

After Jon Hall introduced the Linux track, Frank Hecker, who is a systems engineer at Netscape, gave a talk entitled *The Why and the How* that filled in the background of Netscape's decision to embrace the Open Source model. For those of us who have been involved in what I call “revolution from below”--that is, attempting to sell Linux or Open Source to those in the trenches and let it move up the corporate ladder—Frank had some interesting things to say. He said the engineers didn't believe Open Source could happen with Netscape. So, Frank went to Marc Andreessen with the idea, and the rest is history.

After Frank's talk, we got together for the keynote lunch where Ralph Nader spoke on the Microsoft monopoly and how Linux could be a big player in an alternate solution.

Ralph Nader

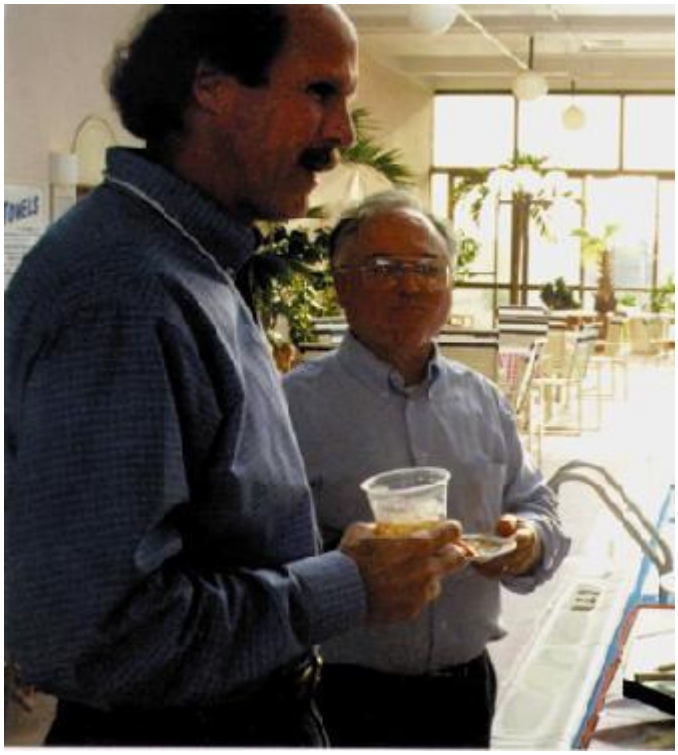
It was a treat to get to meet Ralph and to hear the word Linux come out of his mouth just a few days after hearing Marc Andreessen say it. I brought up how U.S. government procurements used to require a POSIX-compliant operating system be made available with any system they purchased, and Ralph and his staffers are going to look into why that requirement was dropped. I asked him to keep *LJ* up to date on any findings, but you might also wish to check out his web site at http://www.essential.org/.

Two afternoon sessions were held in the Linux track. The first was presented by Ron Workman of Cygnus Solutions, and the second was presented by me. Covering mine is easy—I went through a series of articles which have appeared in our *Linux Means Business* column, illustrating how companies identified a problem, then used Linux to address that problem.

Ron's talk addressed Open Source software from a different direction: how a company can succeed in the business of supporting freely available software. Cygnus was founded in 1989 to provide commercial support for open Internet technologies and has grown to 160 employees. The company has succeeded by offering consulting and support on such products as the Free Software Foundation's GNU C Compiler.

Monday evening was filled with good food and bad beer (Coors Light?—lucky Linus wasn't there) at a beach party hosted by Red Hat Software. Like most parties at these conferences, the evening was filled with meeting others and talking about computers, the UNIX market and telling Microsoft jokes. I found it

a valuable time to get to know people better, for example, Morgan Von Essen and Garry Paxinos of Metro Link, and to thank Alan Fedder and Kathy Goetz of UniForum for organizing the conference.



Ron Workman and Alan Fedder

Tuesday morning started with a Plenary by Eid Eid, President of Corel Computer (see *LJ*, Issue 48). If I had to choose one person who provided me with the most new information, I would choose Eid. I did cheat a little however, as I also talked to him extensively over lunch.

Eid Eid

Eid's Plenary was a pleasant combination of future fantasy and a look at what is available today. The fantasy part consisted of describing a day in the life of Mr. Twenty O'One, illustrating what we should expect in the near future (2001 isn't

that far away) from our personal communicator: voice- and e-mail access, voice commands and an interface to the Corel NetWinder.

At the office, Mr. Twenty O'One will have a typical Intel-based PC but most of his work will be done on his personal communicator and his NetWinder. The NetWinder is much faster for doing common tasks because, while it is a complete Linux system, common tasks (everything from e-mail to video conferencing and replying to e-mail with a voice message automatically appended to e-mail) are trivial to access.

After this fictional introduction, Eid went on to cover the pieces which actually exist today, including the NetWinder, video conferencing, XML and capable portable systems. He went on to say that we need to make all the interfaces and standards open so any vendor can produce pieces which interoperate. I expect some people were surprised when he said, “Corel, Lotus and others will have no choice but to provide some source code for commercial products in the near future.”

Eid also pointed out that Microsoft controls over 50% of the money made on PC applications. Eid was the Chief Technical Officer for Corel Corporation, makers of CorelDraw and owner of Corel Computer before becoming President of Corel Computer. He explained how Corel had been a Microsoft partner until Corel acquired WordPerfect, then Microsoft started hiding technology from them.

As many of us have concluded in the past, Eid believes you cannot successfully fight Microsoft, but went on to point out what you can do. He sees the network computer as the new wave, being less costly and more maintainable. The network computer opens up a new market and, done right, it can become a market as large as today's personal computer market.

The morning session in the Linux Track included a presentation by Don Rosenberg of Stromian Technologies on making money with Open Source as an OEM/VAR. Differing from Ron Workman's presentation of the previous day, Don's presentation was a mini-tutorial on how to get into a market and how to make the right distribution choices along the way.

Next, Morgan Von Essen, President of Metro Link, with the help of Garry Paxinos, Vice President, presented a talk on Metro Link's cooperative work with the XFree86 community—another model of a commercial enterprise working with the free software community. In this case, they developed technology needed for their commercial customers and then gave that technology back to the XFree86 community. Doing this makes it easier for Metro Link to continue development, because their work becomes standard in new XFree86 releases.

Tuesday's keynote lunch was presented by Janpieter Scheerder who was President of Sunsoft and has now moved to head the storage division of Sun. He talked about WebTone and made some very interesting points demonstrating how the non-Microsoft community is growing faster than the Microsoft community. His pointed out that while MS Windows sales are growing at 13% a year, companies like Cisco are growing at 30%, and that while there are over 100,000,000 MS/PC users, that number is a small minority of the six billion people on the planet. He also pointed out that every Quicken user who registers their product is a UNIX user since the Quicken on-line registration server runs on Sun systems.

Janpieter explained that today most ISVs will port their product to either NT or Solaris. By mentioning an old Dutch saying, “the enemy of my enemy is my friend,” he clearly sees Linux as an ally. By the same reasoning, we need to see Sun ports as potential Linux ports of the future.

Janpieter, like Eid, pointed out that the network computer is in—all we need is a $150 device to connect to the Internet. He then went on to talk about two subjects: first “OPEN-standardization” and then Java. This got a reaction from both Eric Raymond and Eid Eid, who both pointed out that for Java to be the answer, its standard needs to be open. There was no resolution, but I am sure we were heard.

The afternoon Linux track brought Bob Young, discussing the size of the Linux market or, more accurately, explaining how hard it is to come up with an accurate estimate. Bob also announced the availability of the *Extreme Linux* CD which includes all the Beowulf RPMs for those of you who want to build your own supercomputer. Jon Hall of Linux International closed the track with some general comments and preparation work for the workshop the following morning.

On Wednesday the Linux track attendees changed their plans and attended the first hour of a presentation entitled *What Exciting Technology is Emerging?* by The Open Group, the organization that owns the UNIX brand and standards. In this presentation, the Open Group announced the UNIX'98 standard.

Historically, vendors such as Digital, Hewlett-Packard and Sun have paid a lot of money to use the UNIX brand. This money goes to support The Open Group's work on the standards and enforcement of the trademark. While it would be great for Linux or a Linux distribution to be UNIX branded, the cost is prohibitive.

The Open Group has now recognized the advantage of getting the Linux community on board. While Sun and others anchor the high end of the Open

Systems community, having an entry-level system compatible with the standard would be a plus. This is not yet a done deal. While the use of the UNIX trademark is limited to those who have paid the fees, the opportunity will be present for a system to be conformant with the standard. Stay tuned, this could be the *in* necessary to tighten the bond between Linux and the rest of the UNIX community.

For more information on the UNIX'98 standard, see the web page http://www.UNIX-systems.org/. Version 2 in HTML can be browsed or downloaded at http://www.UNIX-systems.org/go/unix/.

The melding of minds with regard to UNIX standards, Open Computing and Open Source software needs a venue and UniForum has elected to be that venue. Here's what Alan Fedder, President of UniForum had to say about it:

> UniForum Association is the only forum for open discussion and open debate about open computing. Where else could Eric Raymond and Mike Lambert [of The Open Group] debate each other, listen to each other, and have a better understanding of each other's position? Where else could you hear Ralph Nader, Eid Eid, Janpieter Scheerer, Eric Raymond—all quoting Eric Raymond? Momentous things happened at the UniForum Spring Conference in Ocean City—and I honestly believe that five years from now, 5,000 people will be telling each other they were there when UNIX was saved.

In conclusion, I found attending the conference to be truly worthwhile and look forward to going again next year.

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

Archive Index Issue Table of Contents

Advanced search

Linux Expo a Smashing Success!

Norman M. Jacobowitz

Eric S. Raymond

Issue #52, August 1998

Read all about it...

For three days in May (28, 29, 30), the normally tranquil Duke University Campus was transformed into a raucous playground for geeks and hackers as the Fourth Annual Linux Expo was held at Duke's Bryan Center.

By all accounts, this year's Expo was a smashing success. Red Hat's Marketing Director, Lisa Sullivan, deserves special thanks for organizing and directing the event. Many others, from Key Note Speaker Linus Torvalds to the blue-shirted Duke University catering staff, were instrumental in making it a memorable three days.

According to Sullivan, approximately 1500 visitors were registered as paid attendees, while another 350 to 500 were registered as speakers, VIPs or other gratis attendees. Attendees ranged from as far away as Korea, Finland, Colombia and Alaska. Some 34 exhibitors showed their products and services.

The speakers included:

- Eric S. Raymond gave an inspired, scholarly overview of hacker motivation in his “Homesteading the Noosphere” speech.
- Miguel de Icaza, despite troubles with the overhead projector, shared much about the technical details and future features of “GNOME, The GNU Network Object Model Environment” GUI.
- Mark Mathews described his success as a consultant and Linux programmer in his talk, “Developing Linux Software for Fun—Turns into Profit”.
- Jon “maddog” Hall described his encounters with Linux users worldwide during “Linux Around the World”.

The exhibitors included:

- Corel Computer Corporation displayed their new Linux-based NetWinder Network Computer.
- Digital Equipment Corporation exhibited their latest generation Alpha processors.
- Linux Hardware Solutions showed off some of their line of, well, Linux hardware solutions.
- Caldera, Red Hat and Turbo Linux were there presenting their latest Linux distributions.

Of course, the single most popular event was Friday evening's keynote address by Linus Torvalds. An estimated 1000 to 1200 folks were on hand. In his typically unpretentious, casual and brutally honest style, Linus filled us in on his future vision for the Linux kernel. Linus first took a moment to thank everyone who has helped him with the stable kernel releases, especially Alan Cox. Linus went on to say he is happy with the way Linux is going, especially with the way new markets are opening up and new applications are being made available.

Here are some highlights of Linus's views on important topics for the future of the Linux Kernel:

- The 2.2 release: look for a code freeze in about a month with the next stable release, Kernel 2.2, to follow as soon as late July or early August.
- SMP: Symmetrical Multi-Processing is currently one of Linus's favorite features of the kernel; expect continued development and enhancement of SMP in future releases.
- Merced: Linus is not particularly impressed with or concerned about Intel's upcoming 64-bit processor, code-named Merced—he actually prefers DEC's Alpha architecture. He did say porting Linux to Merced should be no problem once GCC is optimized for Merced.
- Java: While Linus would like to see an officially supported Java Development Kit from Sun, he is still not impressed with Java and would prefer to stay out of the Microsoft/Sun clash over Java purity.
- Emulation: Linus would prefer to see native Linux applications and does not like the idea of emulating other operating systems for the purpose of running applications.

Of course, Linus had much more to say, but the gist of his speech was that with more time and some more good luck, Linux will continue to move towards complete world domination. Judging from the air of excitement and the buzz of optimism pervading this year's Linux Expo, Linus is exactly right.

Editor Wars! by Eric Raymond

Norman M. Jacobowitz (normj@aa.net) is a freelance writer and marketing consultant based in Seattle, Washington.

Archive Index Issue Table of Contents

Advanced search

Evergreen 486 to 586 Upgrade Processor

John Little

Issue #52, August 1998

The kit contains the processor assembly, a chip removal tool, a diskette containing a diagnostic program, an instruction manual and a $10 refund voucher (which, in my case, had expired one week before I actually received the kit).

- Manufacturer: Evergreen Technologies, Inc.
- Email: info@evergreen.com
- URL: http://www.evergreen.com/
- Price: $109.95 ($99.95 with mail-in refund)
- Reviewer: John Little

My wife hates what I've done to the spare room. It has nothing to do with colour coordination or disagreements over wallpaper design. It has everything to do with the fact that almost every square inch of available floor space is given over to the storage of outdated and mostly unused computer equipment. I have everything from an Apple LaserWriter-I to CP/M laptops to a complete 486 tower system stacked in there. Does this sound familiar to anyone?

While that particular 486 tower is too noisy for home use, another 486 system, a desktop Compaq Prolinea 4/33, is one of my favourite workhorses. Originally consigned to the corporate junk heap, I rescued it, swapped the old 127MB disk for a more modern 2.1GB model, installed Linux and then configured it to be the main NFS/Samba server on my home network. In addition to that task, I use it as an xhost server for a Sun IPC (the IPC has a much nicer keyboard and a higher definition display than the Compaq), as a PLIP server for a 386 laptop and also as a compile engine for all of the other 386 class machines in my motley stable. However, while the hour or so required for a kernel compile is much better than the overnight wait required for an i386 machine to complete the same task, it is still fairly tedious, especially when working on experimental changes, so for some time now I've been on the lookout for a viable upgrade.

Unfortunately, bringing yet another system into the house doesn't pass the good housekeeping test. (Besides which, I don't have the cash available for a new system and Pentium Pro machines haven't yet started to appear on the company junk heap.) While a motherboard upgrade looked promising initially, the unique internal layout of the Prolinea, specifically the orientation of the ISA expansion slots, would have meant having to buy a chassis as well as the motherboard. Add the cost of that to my wife's understandable reluctance to allow anything resembling yet another computer into the house, and the obvious answer was to opt for an upgrade processor instead.

A quick check of the local shops here in rural Japan proved disappointing, with a dearth of 486 upgrade options. A couple of shops did carry the standard Intel Over-Drive kit, but at a truly extortionist price. As it happened though, I had

recently received a catalogue from a U.S. mail-order house and a check showed 486 to 586/133MHz upgrades at a very attractive price—only $109.95 for a kit, with the added promise of a much greater performance increase than the Intel version. Was it too good to be true? Well, there was only one way to find out, and at that price I wasn't going to be too much out of pocket even if it turned out to be a complete lemon.

The Kit

The Evergreen 486 upgrade kit comes in a fairly bulky box, with most of the volume being taken up by pre-formed packing designed to prevent damage to the pins on the upgrade processor. The kit contains the processor assembly, a chip removal tool, a diskette containing a diagnostic program, an instruction manual and a $10 refund voucher (which, in my case, had expired one week before I actually received the kit).

The processor assembly consists of a plastic, flat-pack chip mounted on a small PCB with three configuration jumpers to one side. The chip has a heat-sink and fan assembly already attached. There are no external power connections for the fan (operating power being drawn through the socket). The CPU is a 133MHz 5x86 processor from AMD with 16KB of internal cache. The clock speed multiplier is jumper selectable, making the processor a drop-in replacement for 25, 33, 40, 50, 66 or 80MHz 486SX or 486DX chips. The other two jumpers are used to select over-drive or main processor socket placement and the cache mode.

The instruction manual is straightforward, with clear illustrations of the various socket configurations which a typical user might encounter. The jumper settings are well explained, with the notable exception of the cache configuration jumper (more on this later). It also contains lots of sound, common-sense advice, such as marking the orientation of pin-1 of your original processor on the mother board *before* removing it from the socket. (Many motherboards have the silk-screen markings obscured by other components.) In addition, some 22 of a total of 72 pages are dedicated to a fairly comprehensive troubleshooting guide. On the down side (and pretty much as you'd expect), it doesn't contain any mention of operating systems other than DOS or Windows.

This extends to the diagnostic diskette, too. The diagnostic program, **etdiag**, will run only under DOS and, because it is designed to measure system performance, it cannot be run under DOSEMU in Linux. This can be something of a problem in a Microsoft-free zone, but I did discover that etdiag would run under at least two versions of DOS which are readily available from the Internet: one a completely free version and the other a cut-down, evaluation

version of a commercial product—refer to the resources sidebar for URLs. One of the neat things about etdiag is that it will recommend a specific upgrade processor model, based on what it discovers about your system during the initial test, and because Evergreen makes the diagnostic available from their web site, you can use this facility before buying an upgrade kit.

The actual chip replacement process shouldn't cause anyone too much trouble. The chip removal tool supplied with the kit is somewhat thicker than ones I've encountered previously, but with a little bit of patience and minimal effort I was able to ease my original processor out of its socket and replace it with the Evergreen unit. I checked to make sure the fan was turning and the system had completed its power-on self test successfully before replacing the system top cover.

Booting the diagnostic again confirmed the new chip type, higher internal clock speed and increased cache memory were identified correctly. The Dhrystone rating for the CPU (measured under etdiag) had also jumped from the original value of 15384 to a very respectable 39370, increase in performance of two and a half times. Booting Linux showed a BogoMIPs rating increase of exactly four, from 16.59 with the original 33MHz 486 chip, to 66.36 for the 133MHz 586 chip, which is pretty much what you'd expect with a quadrupled clock.

Running Linux

While overall performance is very subjective (few of us use exactly the same mix of tools and applications), I have to say that, for me, the upgrade was a huge improvement. The machine felt completely different with applications starting and running faster and boot-up and shut-down times greatly reduced.

A kernel compile ran almost exactly twice as fast, with the upgraded machine completing a **make clean; make zImage** in 31:36.66 seconds, a task which the original 486 processor completed in an elapsed time (measured using /usr/bin/time) of 1:00:30.24 (slightly over one hour). Given that the rest of the system (main memory and the I/O subsystem) was unchanged, this is quite a respectable increase in performance. Perhaps the biggest perceived change was to the performance of xhosted applications. Netscape, always a CPU and memory hog, started up much faster, as did xterms, a clock and the other applications which I normally run. X performance on the system also improved, but with the current, low resolution monitor and standard VGA video card, I don't envisage using it as a true desktop machine very much (perhaps that's the next upgrade target area). Network (NFS/Samba) performance seemed pretty much unchanged.

The Down Side

There isn't one. Nothing needs to be recompiled or changed in any way. There are no reliability issues. The fan on the Evergreen unit is completely inaudible when the covers are on the system.

The only fault I have with the whole kit is that mysterious cache configuration jumper. It is clearly shown in the illustrations, but ignored in the body of the text. The appendix contains a brief explanation of the differences between write-through and write-back cache. The kit comes with the jumper pre-set to a default of write-through and while it is unlikely anyone could get into trouble using this setting, a simple comment, even if only “leave well alone”, would have been better than nothing at all.

This lack of information on the cache configuration setting led to my trying out Evergreen's technical support by e-mail. They responded to my question within 48 hours, to let me know that the write-back cache option will work only with systems which have been specifically designed with that option in mind. The fact that Evergreen's tech-support responded within a reasonable time and that their web site is an easily accessible, round-the-clock source of information is of no little importance with a product where some degree of “do it yourself” is involved.

Bottom Line

For anyone owning nothing more powerful than a 486 and on a limited budget, this upgrade path is certainly one which I would recommend. Overall system performance has been improved with no decrease in reliability. There were no operating system changes involved and my existing kernel worked fine.

With all of this coming at a cost roughly equivalent to one-tenth of the cheapest, bottom-end system from a mainstream manufacturer, it has to qualify as perhaps the cheapest, legal way to get yourself a “new” machine. Not only that, but your “significant other” need never know.

Resources

John Little , who worked for Sun for nine years, is from the U.K., lives in Japan and works in Tokyo for an American company. He wears a range of increasingly bizarre hats in an (mostly futile) effort to hide his incipient baldness. He can be reached by e-mail at gaijin@pobox.com.

Archive Index Issue Table of Contents

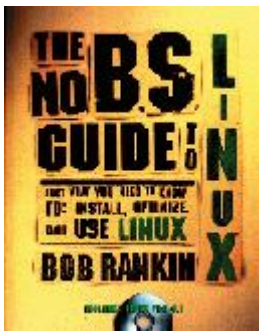
Advanced search

The No B.S. Guide to Linux

Zach Beane

Issue #52, August 1998

The book is bundled with a copy of WGS Linux Pro 4.1a.



- Author: Bob Rankin
- Publisher: No Starch Press
- E-mail: sales@nostarch.com
- URL: http://www.nostarch.com/
- Price: $34.95 US
- ISBN: 1-886411-04-2
- Reviewer: Zach Beane

The No B.S. Guide to Linux is geared to the Linux newcomer who has some DOS and Windows experience. When introducing new concepts or ideas, it relates them to the DOS/Windows equivalents if possible. The tone is conversational and relaxed, even humorous at times.

The book is bundled with a copy of WGS Linux Pro 4.1a. This distribution is based on Red Hat Linux 3.0.3. It comes with several enhancements to the standard Red Hat distribution, the most notable of which is the ability to configure and begin installation from within DOS or Windows. Red Hat 3.0.3 is a few years old, but WGS's philosophy (as described in the CD-ROM's README.TXT) is to use the “greatest, not the latest”.

Unfortunately, I didn't have an opportunity to install WGS Linux Pro. The book deals with several features unique to that distribution, and readers who choose to install WGS Linux Pro will get some very good, detailed help from the book. Chapter 1, for example, gives explicit, step-by-step information to guide a new user through the WGS Linux Pro installation process. This task is difficult to generalize, and the advantage of bundling a specific Linux distribution is evident here. This is in pleasant contrast to some other books I've read where the accompanying CD-ROM seems to be more of an afterthought than a companion to the book.

The book is divided into chapters, each focusing closely on an area of Linux. A typical chapter provides an introduction to a topic, followed by a summary and exploration of tools associated with the topic, and concludes with a pointer to more information on-line or in print.

Chapter 3 of *The No B.S Guide to Linux* covers the requisite introduction to fundamental UNIX concepts such as the hierarchal file system and file name case sensitivity. It also details shell features such as command-line completion, wild cards, pipes and process control.

Chapter 4 introduces a loose collection of utilities useful in the everyday operation of a Linux system. Most of the commands discussed don't have much in common. The chapter covers file and directory manipulation, printing, job scheduling with **at**, printer setup and usage, user listings, examples of the find command and several other indispensable utilities.

There is a cursory look at three text editors (vi, Emacs and pico) in Chapter 5. Although the chapter is fairly short, it gives enough of an introduction to these editors to give you the ability to open a file, navigate through it, make any necessary changes and save it.

The X Window System is discussed in Chapter 7, but only briefly. The chapter provides a working knowledge of how to start X, how to launch some common clients, and how to do some simple customizations. A few more X applications are mentioned in Chapter 11, which discusses the Internet and some popular clients for accessing it. Chapter 7 concludes with what seems to be a slightly out-of-place discussion on shareware, freeware, commercial Linux applications and their locations.

The chapter on programming, disappointingly, breezes through what is one of Linux's most attractive points. A wealth of programming tools are available for Linux, but the author covers only the shell in depth. However, this is in line with a target audience of DOS/Windows users, who may not be interested in using

Linux as a development environment. A few web sites are provided for those who wish to further explore other languages.

A few chapters I haven't mentioned in detail. They cover things from text file manipulation using pipelines of shell commands, to connecting to the Internet and sending e-mail, as well as other topics. In addition, the appendices provide an exhaustive list of the contents of the CD-ROM, the Linux Hardware Compatibility HOWTO and the GNU General Public License.

Someone looking for a weighty reference book won't find it in *The No B.S. Guide to Linux*. What you will find, however, is a readable, thorough and, at times, entertaining introduction to Linux. Although it is a little shallow for the more technically minded, it is well suited to someone apprehensive of being overwhelmed by a totally new operating system. It does a very good job of taking a potentially dry subject and making it interesting and understandable.

Zachary Beane is a programmer for The Maine InternetWorks, Inc. In his spare time he enjoys playing guitar and writing GIMP tutorials. He can be reached at xach@mint.net.

Archive Index Issue Table of Contents

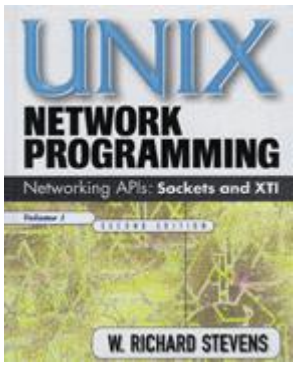
Advanced search

UNIX Network Programming, Volume 1, Second Edition

David Bausum

Issue #52, August 1998

Within several days of working with it, something clicked, and the basics of UNIX network programming fell into place for me.



- Author: W. Richard Stevens
- Publisher: Prentice Hall
- E-mail: sales@prenhall.com
- URL: http://www.phptr.com/
- Price: $59 US
- ISBN: 0-13-490012-X
- Reviewer: David Bausum

In 1990 I chanced upon a review of a book which sounded interesting. I tucked the author's name and the book's title away in my memory, and several months later when I was in a large bookstore, I purchased *Master and Commander* by Patrick O'Brian. Within a dozen pages I was hooked. In the intervening years I've enjoyed all of the 18-volume series dealing with the adventures of Maturin and Aubrey during the Napoleonic Wars. The point of this story is that, when I find an author I like, I'll read as many of his books as I can. I've found the same approach works with technical books.

In *LJ* issue 42, I described how I discovered W. Richard Stevens' *Advanced Programming in the UNIX Environment*. In conjunction with the project described in that review, I needed to learn something about UNIX network programming. By coincidence, Stevens' first book was entitled *UNIX Network Programming*. Without hesitation I ordered the book in late 1996, even though it was six years old and possibly dated. For reasons I don't quite understand (but put down to my own deficiencies), *UNP* and I never quite clicked in the way *APUE* and I did. Even so, when I learned in the fall of 1997 that a second edition of *UNP* was about to be released, I ordered a copy as soon as it was available.

Generally, I am leery of new editions. All too often the book has a new preface, a new cover (with the words “new” or “improved” in big letters), a new copyright date, some cosmetic reshuffling of material, and little else. Some publishers and authors specialize in this game. That is not true for this book. The complete title gives a hint that the new edition is real. (In fact, three volumes are planned for the second edition.) A cursory glance at the table of contents confirms that this is an entirely new book. Within several days of working with it, something clicked, and the basics of UNIX network programming fell into place for me.

I like working with Stevens' books because he makes beautiful books. The operable words here are “makes beautiful”. The second word is important because a beautiful book—one that is well organized and whose pages contain a balanced presentation of text (explaining the topic at hand), diagrams (showing relationships), tables (summarizing or detailing particular items) and code—makes the beginner's journey through new material much easier. The first word is important because a beautiful book does not just happen. It is made in the same way a medieval work of art was made: by an individual who spent years mastering a field, and then used that mastery in various projects. One of the tools Stevens uses is troff. If you have an interest in design and layout, you might profit from the part of Stevens' web site dealing with typesetting and troff resources. Lest the reader think I'm making too much of this, let me note that I'm currently working my way through several books in another area (best left unnamed). Unfortunately, none are as well organized (beautiful in the sense described above) as Stevens' books are. As a result, I find each to be difficult to work with.

The new book has a complete rewrite of *UNP*'s chapters on Berkeley sockets and System V's TLI. Although it is over 1000 pages long, it is not padded and does not (unnecessarily) duplicate material from Stevens' earlier books. There is very little general (non-network) programming covered in the book. References to *APUE* are made as necessary. Similarly, Stevens refers to the TCP/IP series rather than duplicates material covered there. Also, he has managed to cut through the generalities about networking that occupied 10% of the first edition of *UNP*. For example, on page 6 of the new book, Stevens presents code

for a working client program, and on page 13 he presents code for the complementary server. When a client connects, the server gets the time and sends it to the client which prints it and quits. The result is trivial (just as the classic “hello world” program is), but it provides a working client-server program hardly a dozen pages into the book.

This new book mixes working, portable, real-world, annotated code with text, diagrams and tables to document the sockets (and XTI) API. Over 50 programs and 100 functions are discussed in the book.

Working code means you can obtain all the examples used in the book from Stevens' home page (http://www.kohala.com/~rstevens/). For example, in less than an hour I downloaded a 226KB gzipped tar file, expanded it, ran Stevens' configuration routine, ran the Makefile which builds the library used by all programs and subsequent Makefiles, and compiled the programs for Chapter 1. During the make of the library, I had a small problem because I don't have a threads library on my system. A quick change to the library Makefile (removing references to threads) fixed things. I've not compiled all the programs supplied, but the ones I have compile and run the way the book says they should.

Portable means the code will compile not only on Linux systems (yes, Linux is one of the systems used as an example in the book), but on BSD, HP UNIX, Digital UNIX, Solaris (Sun/SPARC) and Unixware systems. Some of these are running IPv4 protocol stacks, others are running IPv6. The code “doesn't care”; it is portable. One of the themes running through the book is how a programmer can write library routines which are protocol stack and system independent. Portable also means POSIX compliant, in particular Draft 6.6 (March 1997) of P1003.1g.

Real world means the code deals with errors. A programmer must constantly ask “how can this code, this library call, this function fail? If it does, how can I either gracefully end the program or recover and keep going?” For many new programmers this is a difficult lesson to learn. Stevens worries about this, and the beginner will benefit from his examples.

Annotated means that when code is discussed in the book, the printed listing includes the subdirectory and file name for the code, the lines in the listing are numbered, and the discussion in the text puts an appropriate, line-number range in the margin of the text. This makes it easy for you to move from text (discussing a range of lines) to the printed listing (in the book) to the actual file (on your hard disk).

The book contains much more than code. Stevens writes well. He has developed a style which allows him to interrupt his narrative discussion with

historical and current observations. He does this by adjusting the margins and changing the font. It is easy to skip the “notes” on a first reading and then focus on them on subsequent passes through a section or chapter. One example of the timeliness of these notes is his explanation of *denial of service* attacks on page 99. When appropriate, Stevens includes a diagram which provides an alternate explanation for a topic. Also, when appropriate, he includes a table which summarizes similar pieces of a topic. The result is a work which is encyclopedic without being stifling.

I find I can approach a new chapter or section and first study the code, then examine the diagrams, then read the text and finally repeat the process, possibly in a different order. As I understand more, the various approaches reinforce each other so that the whole ends up being greater than the sum of its parts. Finally, the various tables are natural places to refer back to, once I understand a topic or when I have a specific question.

The book is organized into four parts. The first is 50 pages (two chapters) of introductory material. This includes the working client-server program referred to above. The second is 200 pages (seven chapters). This discusses the basic networking functions utilized by Berkeley sockets (socket, bind, etc.). This part contains many programs and program fragments. The third is 500 pages (18 chapters) and deals with miscellaneous topics. The fourth is 120 pages (seven chapters) and covers the basic API for X/Open Transport Interface. The beginner will want to read the first two parts carefully. The more advanced reader should skim part one to get a sense of Stevens' style and then refer to part two as needed. To a large degree, the chapters in part three are independent of each other. That means, once the basics from part two are clear, it is very easy to pick and choose among the topics discussed in part three. If you are interested in threads, skip to Chapter 23. If you are interested in IPv4 and IPv6, turn to Chapters 10 and 24.

Rather than give more details here, I will invoke the spirit of Linux and refer you to Stevens' web site. If you look there, you will find information about each of Stevens' existing books, and you will find the Table of Contents, the Preface and a sample chapter of *UNIX Networking Programming, Volume 1, Second Edition*. In addition, you will find the code for the book and a README file describing the installation of the software, as well as the typesetting and troff references. Finally, you will find information about Volume 2 (*Interprocess Communication*) and Volume 3 (*Applications*). Both these volumes were still under construction at the time I wrote this review (February). Together the three volumes comprise the second edition of *UNIX Network Programming*.

David Bausum received a Ph.D. in mathematics from Yale in 1974. Since the early 80s, most of his energy has gone into software development and related

activities. He coedited The Journal of Military History Cumulative Index: Volumes 1-58, 1937-1994. He can be reached via e-mail at davidb@cfw.com.

Archive Index Issue Table of Contents

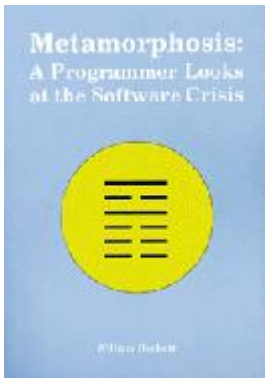
Advanced search

Metamorphosis: A Programmer Looks at the Software Crisis

Harvey Friedman

Issue #52, August 1998

Could this possibly be a Kafkaesque treatise on debugging?



- Author: William Beckett
- Publisher: Numerical Analog, Inc.
- Price: $15 US
- ISBN: 0-966033-9-6
- Reviewer: Harvey Friedman

One of the reasons I agreed to review this book was that I was intrigued by the title. Could this possibly be a Kafkaesque treatise on debugging? As I read it, I realized that such was not the case. I found the style more Conradian, as in *Heart of Darkness*.

The book has three main sections. The first, comprising 9 chapters, is a response to an article in the September 1994 issue of *Scientific American* titled "Software's Chronic Crisis". The author, William Beckett, is an experienced systems designer/programmer who has used many programming languages on many hardware platforms. The writer of the *Scientific American* piece appeared to be without an extensive computer background, so drew much of his article

from management apologetics and computer shock headlines in the popular press.

Beckett responds section by section, paragraph by paragraph, and even line by line in these nine chapters. His theme is common sense or pragmatism. Non-entrepreneurial programmers will enjoy this part of the book because it is the working stiff talking back to “Dilbert”-style management.

Section 2 comprises chapters 10-15. Here, Beckett reprints and discusses letters and responses from Paul Allen and from *Wired*. He uses lots of personal anecdotes illustrating points he wants to emphasize. This too is a good read, particularly for the disenchanted.

Section 3, chapters 16-21, is where things become highly speculative. One could say that here we have a melange of science fiction, hermeneutics, New Age thoughts, etc. Section 3 should be called “the world according to Beckett, and how Beckett would make it better”. I found myself slogging through the final section.

Overall, this book could provide useful reflection for programmers who are not so overworked that they have time to actually read and think about his controversial ideas. The avowed purpose of the book “is to provide professional men and women in creative fields with a perspective on current Western ideology which is capable of transforming it for the better”. He supports it with an eclectic bibliography of 54 items for further reading. This book will not be everyone's cup of tea by any means, but some might find a few good ideas or explanations.

Metamorphosis can be purchased through amazon.com, or write the publisher at P.O. Box 631, Snohomish, WA 98291 for more information.

Harvey Friedman is a computer consultant at the University of Washington, functioning either as system administrator or statistical analyst. In his leisure time he likes playing with Linux and enjoys orienteering, the sport of navigation. He can be reached via e-mail at fnharvey@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

Creating Web Plots on Demand

Mark Pruett

Issue #52, August 1998

Mr. Pruett tells us how his company creates on-the-fly plots of database information for web display.

I work at Virginia Power, an electric utility that services a large section of the southeastern United States. Part of my group's job is to collect and archive data from thousands of devices in the field, such as electrical transformers and circuits. Our use of Linux to collect this data is a story in itself (see “SCADA—Linux Still Hard At Work” in the January and February 1995 issues of *Linux Journal*).

To make this vast amount of data accessible throughout our company, my group built a set of Intranet applications. These applications are accessible to anyone on our company network using a standard web browser.

Using one of these applications, our users can generate graphic plots of data from our databases. For example, an engineer may want to see a plot of the electrical load on a particular transformer during the course of a month. Our program can extract data from our database, create a plot of that data and display the graphic through the user's web browser in a matter of seconds.

Using tools that come with most standard Linux distributions, you too can build web-based plots on demand. The tools you will need include Perl, the **gnuplot** graphics package, and the **NetPBM** graphics conversion tools. I'll assume you already have a web server up and running. We use the Apache web server, but the techniques described here will work with other web servers too.

Before I show you how to tie these tools together, you'll need to get and install gnuplot and the NetPBM packages. See “Resources” for information on Internet sites and Linux distributions where you can get these packages.

What is gnuplot?

gnuplot is a device-independent plotting package that has been available for various Unices, as well as other platforms, for many years. It features its own language for creating and displaying two- and three-dimensional plots. Commands can be entered interactively, one at a time, or can be placed together into a gnuplot script file and run as a batch job.

gnuplot supports a wide variety of output devices. The most useful of these devices can be an X window. You can run gnuplot from an xterm under X Windows, and it can plot your data into a separate X window. This is a good way for those unfamiliar with gnuplot to learn all about the various commands.

Since gnuplot can take its commands from script files and its data from simple text files, it's the perfect tool to automate the generation of graphics when requested by a web server.

Plotting access_log Data

I doubt you have much interest in plotting load curves of electrical transformers, so I'll present a more generally useful example. The example is a CGI Perl script that plots web server hits over the course of a given day.

The script will require some understanding of Perl; the small amount of CGI knowledge required will be explained as needed.

First, let's describe what the script needs to do. Every time a person requests information from an Apache web server, the request is logged to a file called **access_log**. This log resides in the web server's log directory. The exact location of this directory depends on how you have set up your web server. Each line of the log represents a single web hit. A line from an Apache server's **access_log** file looks something like this:

```
foobar.mydomain.org
- - [01/Sep/1997:17:14:31 -0400]
"GET /images/gnuplot_10270.gif HTTP/1.0" 200 9538
```

This line indicates that a web browser at foobar.mydomain.org requested the file /images/gnuplot_10270.gif to be sent to it at 5:14 PM on September 1. All lines in the log have the same format, so we should be able to extract just those lines that match the date we're seeking.

We can also tell the hour and minute of the access. We'll want to keep a count of the number of accesses for each minute of our target day. This data will be sent to gnuplot so it can create our plot. We'll want to create a simple x-y line plot of the data, with time of day on the x axis and number of hits on the y axis.

Once the graph has been plotted, we'll need to convert it to a graphics format that can be displayed with a web browser. Finally, we'll need to build an HTML page to send back to the browser. This page will have an image tag in it that points back to the graphics file we just created.

Don't worry if this seems like a lot of hoops to jump through. It can all be done in seven short steps with less than 100 lines of code. The complete Perl script is shown in Listing 1. The code is divided into numbered sections for easy reference. Let's look at the script one section at a time.

Section 1: Where to Find the Files

The first section of Listing 1 requires little explanation. I just assign variables for file names and directories that will be used later. This is the section you might change if your web server's directories are different from mine.

Section 2: The Date to Search For

The Perl script is meant to be started when a web browser requests it. In other words, it is a CGI program. A Common Gateway Interface (CGI) program provides a standard mechanism for providing "interactive content" over the Internet. Instead of requesting a static HTML document, the web browser asks the web server to run a program that will dynamically create an HTML document. CGI programs can be written in any language, but Perl is by far the most common language for CGI programming today.

My CGI script will actually run as a stand-alone program from the command line, although it isn't particularly useful when run that way. Running as a CGI program, it gains one important aspect: access to the **QUERY_STRING** environment variable. The **QUERY_STRING** environment variable is one way to pass information to a CGI program. If **QUERY_STRING** has been defined, then our program expects it to contain a date. This date should be in dd/mmm/yyyy format (e.g., 12/Sep/1997). That's the same format Apache uses to log its dates.

If **QUERY_STRING** isn't defined, we'll assume we should look for access data for today's date. In either case, the date we're seeking is stored in the variable **$date**.

Section 3: Searching the access_log File

In section 3 of Listing 1, we open the Apache access log and begin looking for lines that contain our target date. When we find a date that matches, we extract the hour and minute and place them in the variables **$inhour** and **$inmin**.

It's at this point that we run into a slight conceptual problem. We want to plot time of day on the x axis of our plot, but what we actually have is hours and minutes. What value will be plotted on the x axis? What we want is a time line from 12:01 AM to midnight. If we just turned times into integers, we would get values like 1024 for 10:24 AM. **gnuplot** could plot this, but since there are only 60 minutes in an hour, there would be a gap at the end of each plotted hour. There are many ways to solve this problem. The one I used was simply to scale the minutes 0 to 59 to a value between 0 and 99. The hours are then multiplied by 100 and the converted minutes are added to them. Thus, the time 1:30 PM becomes the integer value 1350. Our plot will only have tic marks for hours, so no one will notice our little conversion.

So in this section of code, we convert the hours and minutes to an integer value and use it as an array index for an array of counters called **$accesses**.

Section 4: Getting the Data Ready for gnuplot

gnuplot can plot the data it reads from a text file. We need to write a file for gnuplot, where each line of the file contains two integer values. The first value will be our time value (the x axis) and the second value will be the number of web hits at that time (the y axis). The file will contain 2401 values. Here are a few lines from an example file:

```
0808  1
0809  0
0810  1
0811  2
0812  0
0813  21
0814  0
0815  12
```

This file shows a few minutes during the 8 o'clock hour. We create the file by printing out the data in our $accesses array. Here we encounter another little “gotcha”. Perl creates something called “sparse arrays”. This just means that array elements are created as they are defined. In other words, if there were no accesses at 8:12 AM, as in our sample data above, then no corresponding array element is created. It's undefined.

Normally in Perl, you can traverse an array using a **foreach** loop. In our case though, we need to output data for all time intervals, even those times when no accesses took place. We need this so we have a contiguous set of data for gnuplot to graph.

This is actually very easy to do. In section 4 of Listing 1, we set up a simple **for** loop. Using Perl's defined function, we determine if an array element exists. If it does, we simply print the index value as our x value and the access count as

our y value. If the array element isn't defined, we know there were no accesses at that time, so we print out a zero as our y value.

Section 5: Building a gnuplot Command File

gnuplot will automatically label the x and y axes with values from our data set. If the maximum number of web hits for any time interval in the data set was 48, then gnuplot might create tic marks on the y axis at 0, 10, 20, 30, 40 and 50. This would be fine for the y axis, but remember that the x axis contains a calculated time value that might not make much sense to a person reading our graph.

Luckily, we can override the default tic marks and create our own. That's the job of the **for** loop at the beginning of section five. We create a text string that will be embedded in the gnuplot command file we are about to create.

gnuplot creates plots based on a set of commands you provide it. Actually, gnuplot has only two commands for plotting data, **plot** and **splot**, and we'll be using the simpler of the two in our program. The other command we'll use is **set** to enable and disable particular options and features in gnuplot. The Perl print statement in section five of Listing 1 handles writing a gnuplot command file to disk.

Those not overly familiar with Perl may find this variation of the print statement somewhat confusing. Let's look at that line:

```
print GPFILE <<EOM;
```

This print command tells Perl to write every line in the Perl script following it to the file opened with the GPFILE file handle. It stops printing lines to the file when Perl encounters the line in the script consisting only of the letters EOM. So all the lines in section five following the print statement and continuing to the EOM line are not Perl statements at all, but are gnuplot commands that will be written by Perl to the gnuplot command file.

The print command will also substitute the Perl variable references with their values, so the line:

```
set title "Web Server Accesses $mon $day, $year"
```

might be written to the file as:

```
set title "Web Server Accesses Aug 12, 1997"
```

Section 6: Creating the Plot Graphic

Now that we have a data file and a gnuplot command file, how do we get our plot? And how do we show it to the user? This turns out to be the easy part.

First, we need to talk a bit more about gnuplot. I mentioned earlier that gnuplot can display to a variety of devices, including X terminals. But gnuplot can also write a file in portable pixmap format (PPM). In section five of the Perl script, note that we write the gnuplot command:

```
set term pbm color
```

to the gnuplot command file. This tells gnuplot to write the output to a file.

However, this only takes us part of the way. Web browsers don't know what to do with PPM files; they usually want either a GIF or JPEG file. That's where the NetPBM package comes in. This is a set of command line utilities that convert from one format to another. One of those tools is exactly what we're looking for: the aptly named **ppmtogif**. **ppmtogif** is a simple filter program. It takes a PPM image file from standard input and writes a GIF file to standard output. Since most web browsers support GIF, ppmtogif fits our needs perfectly.

Thus, section six turns out to be almost trivial. We use a Perl system call to run our UNIX commands, then run gnuplot using the command file we created in section five (which in turn will read the data file we built in section four). **gnuplot** sends the graphic in PPM format to standard output, and the ppmtogif filter turns it into a GIF file that is redirected to a disk file. We now have our graph in GIF format on disk.

Section 7: Displaying the Plot in a Browser

The final step (section 7 of Listing 1) is to display the graph in the user's web browser. Remember this is a CGI program, so the browser that invoked the CGI Perl script is waiting to receive something from our web server. What we'll deliver is a brief HTML page containing an HTML image tag that points to the GIF file we just created.

We use the same technique, a multi-line print statement, to create the HTML the browser will receive. Note that we must prepend a “content-type” preamble to the data we send back to the browser. This lets it know to expect an HTML page, rather than some other type of data.

The user generates the plot by typing in the URL of the CGI script in their browser. If the server is called **myserver** and our script is saved as **usage_graph_cgi** in the web server's cgi-bin directory, then they would type:

```
http://myserver/cgi-bin/usage_graph_cgi
```

To specify a date other than today, the user appends the date to the end of the URL, separated by a question mark:

```
http://myserver/cgi-bin/usage_graph_cgi?11/Sep/1997
```

If all goes well, and there's no reason why it shouldn't, the user should see a page that looks something like Figure 1.

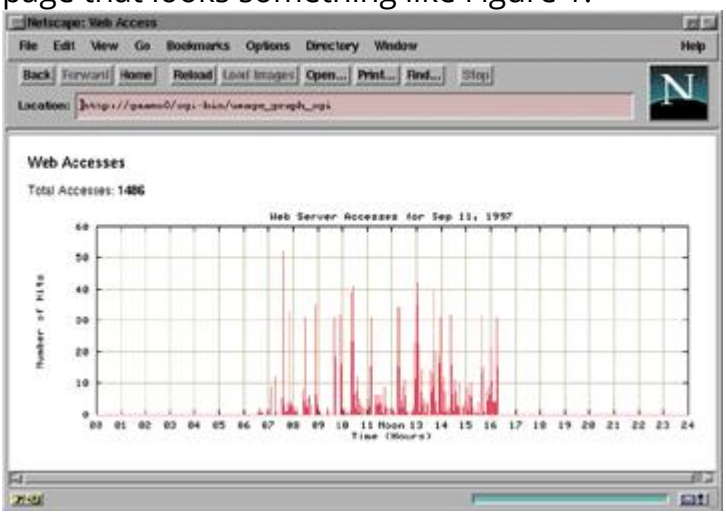


Figure 1. Plot Displayed on the Web

Cleaning Up the GIF Files

One minor issue remains. We create a new GIF file each time our Perl program is run, so we must get rid of them when they're no longer needed.

In my case, each night I simply run a script under crontab that deletes any GIF files created during the day. Since most of our applications are run during the daytime hours, the chances of deleting a GIF file I still need are very small.

This scheme might not be acceptable in every situation, so you may need to devise a different way to clean up the GIF files that collect on your web server. [The **find** command will fit this purpose admirably—Ed]

Reviewing the Steps

While I've provided a specific example of on-demand plotting, the techniques used can be applied to any type of data you might want to plot. If you can extract the data to a simple text file, and if the data lends itself to two- or three-dimensional plotting, you can deliver it to the Web. The basic steps are always the same:

- Build a text file with the data to plot.
- Build a gnuplot command file.

- Run gnuplot to build the plot in PPM format.
- Convert the plot to GIF format using ppmtogif.
- Build an HTML page with the image tag and send it to the browser.

Tying together tools like gnuplot and NetPBM to quickly build a useful program shows that software doesn't necessarily have to come packaged in the latest object-oriented component, tied together with ActiveX or CORBA. Often, good solid tools, text files and a touch of Perl will more than suffice to do the job.

Resources

Mark Pruett received his M.S. in computer science from Virginia Commonwealth University. Mark is a programmer who writes about programming. He hopes some day to be a writer who writes about how to write program documentation. He can be reached at pruettm@vancpower.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Speeding up Database Access with mod_perl

Reuven M. Lerner

Issue #52, August 1998

Continuing the discussion of mod_perl, Mr. Lerner tells us about the DBI specification and the Apache::DBI module.

Last month, we started to look at **mod_perl**, a module for the Apache web server that puts a copy of Perl inside of our HTTP server. Not only does mod_perl save us the overhead of forking a new process and invoking Perl each time a CGI program is run, but it caches programs once they are compiled, reducing start-up time even further.

As with everything else in life, there are trade-offs for using mod_perl. CGI programs can be written in any language and can be run by any web server on any platform. By contrast, mod_perl only works with Apache (which runs under most UNIX versions and is about to be released for Win32), and requires that programs be written in Perl. If you are interested in maximum portability, you might consider sticking with CGI. But if you are like me and spend most of your time working with Apache and Perl anyway, you might seriously consider moving toward mod_perl.

However, this is not the end of the story. Even if mod_perl were infinitely faster than CGI's fork-and-execute method, a major bottleneck would still remain: opening a connection to a database server.

If your web application talks to a relational database server, it has to open a connection before it can send a query. That initial dialogue between your program and the database server can take quite a while, both because of the initial TCP connection that must be established and because database servers were not designed for single-query connections. Many servers expect you to connect once, send a number of requests, receive a number of responses and disconnect when done. Web applications, by contrast, tend to open a connection for each query, which can unnecessarily slow your programs.

Luckily for us, the folks who wrote mod_perl created a module designed to solve this problem. **Apache::DBI**, as it is called, takes advantage of mod_perl's variable persistence (i.e., the fact that variable values are kept from one invocation of a program to another) to keep database connections open. Each invocation of a program using Apache::DBI has only to send a query to the database and act on the returned response.

This month, we look at the DBI specification in general, and the Apache::DBI module in particular. We also take a quick look at the **Benchmark** and **LWP** modules, which can help profile code in general, and which will allow us to see just how much faster mod_perl and Apache::DBI can make our programs.

Configuration

To get mod_perl working with Apache, you will need to compile and install the latest versions of both programs. (Full instructions are available in last month's "At the Forge".) To take advantage of all of Apache::DBI's features, you will need to compile mod_perl with more than the default options. These options can be specified by adding **OPTIONNAME=1** on the same line as the initial invocation of Makefile.PL, the first stage of the compilation and installation process.

I found it easiest (if a bit wasteful) to compile using the **EVERYTHING** flag, which turns on all of the mod_perl options, even those that are unnecessary for Apache::DBI. To do this, type the following in the initial mod_perl directory:

```
perl Makefile.PL EVERYTHING=1
```

The rest of the mod_perl and Apache configuration continues as described last month. See the INSTALL document that comes with mod_perl to learn how to turn on only those features you need, rather than using **EVERYTHING=1**.

Apache::DBI works automatically with programs that use mod_perl. In other words, any program that uses Apache::Registry (more or less the mod_perl equivalent of CGI) automatically gets the benefits of Apache::DBI, assuming that the latter is specified in the configuration file (described below). You can configure a directory to use Apache::Registry in much the same way as you can configure it to use CGI, with directives in the srm.conf file such as:

```
# Deal with mod_perl
<Location /perl-bin>
SetHandler perl-script
PerlHandler Apache::Registry
Options ExecCGI
</Location>
```

You can then instruct mod_perl to load Apache::DBI in all directories using Apache::Registry by inserting the following directive into srm.conf:

```
PerlModule Apache::DBI
```

Now you will need to restart your server, most likely while logged in as root, by typing:

```
# killall -v -1 httpd
```

The server is now ready to talk to the database using mod_perl. Before we can take advantage of Apache::DBI, though, we will need to investigate DBI a bit more carefully.

What is DBI?

There are dozens (perhaps hundreds) of databases on the market, some of which run under Linux. One product which I have used, both in my own consulting work and in the pages of this column, is MySQL, a “mostly free” database (in the author's words) distributed by TcX DataKonsult AB. The “Resources” sidebar contains information on where you can download the source and binaries for MySQL.

The CGI programs we wrote in our previous encounters with MySQL used the **Mysql** module for Perl, which gives us access to all of MySQL's features. Mysql.pm continues to work just fine for most applications.

However, if you are interested in keeping up with the latest standards and trends within the Perl community, you should switch (as I have) to DBI, the generic database interface for Perl programs. DBI allows you to use the same code on any number of databases. That is, you can write a program that talks to MySQL, Sybase, Oracle or any other database product—and you will have to change only one word in order to port the program to another database product. This makes Perl a very powerful and portable database-access language.

DBI is divided into two parts: the generic **DBI** module engine, which can be downloaded from CPAN, the Comprehensive Perl Archive Network, and a DBD (database driver) for the particular brand of database you wish to access. There is only one DBI module, but there is a different DBD for each database you might wish to access. (See “Resources” for information on where you can obtain DBI and DBDs.) If you plan to use more than one database server, you will have to install more than one DBD. You can install as many DBDs as you want; they are installed in parallel, and thus do not conflict.

If you are using MySQL, then you will have to download the driver for Msql, the database on whose interface the MySQL API was modeled. Before it compiles and installs the necessary modules, the **Msql-modules** package asks whether you want to install DBDs for Msql, MySQL or both.

Using DBI

Once you have installed DBI and the appropriate DBD, you will be able to do just about everything you would normally expect from a database. The syntax is a bit different from the syntax we have seen in previous installments of "At the Forge", but is conceptually quite similar. It should not take you very long to start using DBI, once you have seen some examples.

Connections to the database are kept in a database handle, normally stored in a variable called **$dbh**. The database handle not only gives you a compact, object-oriented way to access database methods, but also means that you can connect to multiple databases at the same time, giving each connection its own database handle. (This might be useful when moving information from Sybase to Oracle, for example.)

The basic syntax is fairly straightforward:

```
$dbh = DBI->connect($data_source, $username,
        $password);
```

As you can see, the **connect** method takes three arguments. The first, **$data_source**, defines the database you wish to access, as well as the name of the computer on which the server sits and the access port on that computer. The second two arguments are theoretically optional, but most configurations will (and should) require them.

For example, most test programs on my home computer use the following syntax:

```
$dbh = DBI->connect("DBI:mysql:test:localhost");
```

Because I use the unprotected "test" database, no user name or password is necessary. A production site on which user names and passwords are required would use syntax like the following:

```
$dbh = DBI->connect("DBI:mysql:classifieds:dbserver",
        "classy" "51haf3");
```

In the above line of code, we are again connecting with the MySQL DBD. But this time, we are connecting to the database named **classifieds** on a machine named **dbserver**, with the username **classy** and password **51haf3**. Remember that user names and passwords on database systems are unrelated to user names and passwords on UNIX systems. For security purposes, you should use different passwords (and perhaps even different user names) with your databases than are actually in use on your system.

If the connection succeeds, **$dbh** can be used as an entry point into the database. If the connection fails, **$dbh** remains undefined. This allows us to use the following error-checking code:

```
&log_and_die($DBI::errstr) unless $dbh;
```

The **&log_and_die** routine is one of my old favorites (and probably familiar to long-time readers of this column), printing an error message on the screen and then exiting gracefully. Complete listings including the subroutine &log_and_die are available at ftp://ftp.linuxjournal.com/pub/lj/listings/issue52/2991.tgz.

Now that we are connected to the database server, we can feed it one or more queries in SQL, Structured Query Language.

If we want to insert a value into a table in the database, we can simply say something like:

```
$sql = "INSERT INTO test_insert (contents) VALUES
(\"$random\") ";
```

Putting the SQL query into a scalar variable before using it is not required, but helps if and when you need to debug the code. (This way, you can easily add a "print" statement in the middle of your program.)

With the query all set, we tell the database to perform the requested action using the **do** method. This returns a variable which I call **$successful_insert**; much like **$dbh**, **$successful_insert** is defined only if the query was successful:

```
$successful_insert = $dbh->do($sql);
print "<P>Success!</P>\n" if $successful_insert;
```

Finally, we disconnect from the database. This is not completely necessary, since Perl will close connections when we are no longer using them. Nevertheless, it is always a good programming practice to clean up:

```
$dbh->disconnect;
```

The above syntax is good for any SQL query from which we do not expect a result, namely **INSERT**, **DELETE** and **UPDATE**. (For more information about SQL, see Resources for some book recommendations.)

Retrieving Rows with SELECT

If we want to retrieve matching rows from the database, we need to modify the syntax just a bit. After all, we expect to not only receive a report on whether the database was able to perform our requested action, but also see the results.

Assuming we are connected to the database, we set **$sql** to our SQL query:

```
$sql = "SELECT id,contents FROM test_insert";
```

We then use the **prepare** method to send our query, as follows:

```
$sth = $dbh->prepare($sql);
```

The **$dbh->prepare** result is known as a "statement handle", which is traditionally named **$sth**. Just as **$dbh** allows us to perform operations on the database to which we have connected, **$sth** allows us to perform operations on the statement we have just sent. And just as **$dbh** is undefined in the case of an error, so too is **$sth**:

```
&log_and_die($sth->errstr) unless $sth;
```

Assuming that **$sth** was sent to the database successfully, we tell the database to execute our query, checking for problems with the return code:

```
$sth->execute || &log_and_die($sth->err);
```

Now comes the fun part, namely iterating through each of the rows returned to us. We can find out how many rows were returned as a result of our query by checking the value of **$sth->rows**. We can then retrieve each of the returned rows, one by one (with one column value per row), using **$sth->fetchrow**. When there are no more rows to retrieve, **$sth->fetchrow** returns false, which means that we can use it within a "while" loop. Indeed, this is a fairly standard idiom in the DBI world:

```
# Loop through returned rows
while (@row = $sth->fetchrow)
{
# Grab the columns from the row
$id = $row[0];
$contents = $row[1];
# Print the ID and the contents
print "<P>$id:\"$contents\"</P>\n";
}
```

When we are finished with this statement, we use the **finish** method associated with the statement, which is analogous to the **disconnect** method for the database handle:

```
$sth->finish;
```

Now that we have reviewed all of this in theory, let's put it into practice. First, we will create a small table in the "test" database in MySQL, by running the **mysql** client program:

```
mysql test
```

Once we see the **mysql>** prompt, we can create our small test table:

```
CREATE TABLE test_insert
 (id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY
        KEY, contents VARCHAR(50) NOT NULL,
        UNIQUE (contents));
```

The above defines our table, **test_insert**, to have two columns. The first column, **id**, is defined to contain an unsigned integer. The integer, whose presence is mandatory (**NOT NULL**), is automatically incremented every time we insert a row into the table and can be used as a unique index into the table. The second column, **contents**, is a variable-length character string whose presence is mandatory (**NOT NULL**) and which cannot be repeated in another record (**UNIQUE**).

The CGI program in Listing 1 demonstrates all of the above, first inserting a number of rows into the table and then retrieving them. Most DBI programs are as simple as this one, although many either store or retrieve information, rather than do both.

One of the problems with a standard like DBI is that the interface follows the least common denominator. That is, there are differences between database packages in addition to their administration and speed; just about every package includes a number of non-standard SQL commands and features in order to differentiate itself from the competition. If you are interested in using such features, you might have to use the **func** DBI method, which enables proprietary database extensions. Of course, doing so means that your program is no longer portable to other databases, which might be a concern if you switch to another vendor.

Moving the Program to Apache::DBI

We have now compiled Apache to use mod_perl, configured a perl-bin directory for serving mod_perl programs and configured Apache to insert the Apache::DBI module for all programs within the perl-bin directory. We are all set to take our sample DBI program and use it with mod_perl.

How must we modify the program in order to get it to work with mod_perl? Actually, we needn't make any modifications at all, if we have configured our copy of Apache as described above. All we need to do is copy our program into perl-bin, set the appropriate permissions and give it a whirl. Here's what I wrote on my computer, for example:

```
~httpd/cgi-bin% cp dbi-demo.pl ../perl-bin/
~httpd/cgi-bin% chmod ug+x ../perl-bin/dbi-demo.pl
```

I changed the URL in an open browser window such that it pointed to perl-bin rather than cgi-bin, and—voilà--it all just worked.

When I first started to use mod_perl and Apache::DBI, I wasn't sure how much faster programs would run. The execution certainly seemed faster, but I wasn't sure how much of an improvement I was seeing. I decided to use Perl's Benchmark module, comparing the execution speed of two different programs.

I would try to insert 100 random text strings into a database, first using a CGI program and then using an Apache::DBI version of the same program (which, as we now know, simply means a version of the program placed in the perl-bin directory).

Benchmarking is a tricky and subtle business, and there are undoubtedly factors which I neglected when calculating these results. Even so, they seem to bear witness to the amazing performance difference between CGI and mod_perl. I'm sure if I were to spend a great deal more time working on my Apache and/or MySQL configuration, I could get even better performance out of my lowly 75 MHz Pentium running Red Hat 4.2. However, the relative numbers should speak for themselves.

First, let's examine the test I performed. I used the same test_insert table in MySQL as we saw before. I then wrote a CGI program, similar to the one we saw before, which connects to the database and inserts a random value into the **contents** column. The resulting program is shown in Listing 2.

How Fast is it?

Now that we have our test program, how can we actually test it? I wrote a short Perl program using Benchmark.pm, in Listing 3. The **timethese** function is imported by Benchmark.pm, which we bring in at the beginning of the program. We also bring in **LWP::Simple**, part of the "Library for WWW access in Perl" that makes it a snap to write a small web client. How simple? Well, the following one-liner returns the HTML contents at http://www.linuxjournal.com/:

```
perl -e 'use LWP::Simple;
print get "http://www.linuxjournal.com";'
```

Perl does not format the output for you. That's the difference between a web browser and a web client; the former is meant to retrieve information for humans, while the latter is meant to retrieve information for programs. In this particular case, we just want to simulate 100 retrievals of each of our programs via the web. Any timing differences will thus be due to the program on the server side—which, since they are identical, means that the differences will be due to mod_perl and Apache::DBI.

How much faster is the Apache::DBI version than its CGI counterpart? Here are the results I got running time-db.pl:

```
[1086] ~% ./time-db.pl
Benchmark: timing 100 iterations ...
Apache::DBI: 24 secs (1.77 usr 0.67 sys = 2.44 cpu)
Plain CGI: 394 secs (1.10 usr 0.61 sys = 1.71 cpu)
```

That's quite a difference. When I first ran this benchmark, I was convinced that the plain CGI program had somehow gotten stuck. Alas, that was not the case; the overhead associated with CGI was simply too great.

Here is a second run of the same benchmark, just for comparison.

```
[1099] ~% ./time-db.pl
Benchmark: timing 100 iterations ...
Apache::DBI: 28 secs (1.89 usr 0.61 sys = 2.50 cpu)
Plain CGI: 355 secs (1.15 usr 0.62 sys = 1.77 cpu)
```

Yes, it looks like CGI is indeed much slower. By the way, you can see that Apache::DBI used more CPU time than plain CGI—which means that the time was spent forking the new Perl process, rather than performing our program's computations.

What if we take out the Apache::DBI directive in srm.conf and restart the server? That would give us an indication of how much overhead was being used opening the database connection. As you can see, things do indeed slow down—although admittedly not by a huge amount:

```
[1104] ~% ./time-db.pl
Benchmark: timing 100 iterations ...
Apache::DBI: 34 secs (1.97 usr 0.63 sys = 2.60 cpu)
Plain CGI: 460 secs (1.19 usr 0.60 sys = 1.79 cpu)
```

The moral, then, seems to be that moving from CGI to mod_perl gives a huge performance boost, and that moving from DBI to Apache::DBI gives a moderate performance boost. The more database accesses your web applications do, the more useful these technologies will probably be in your work. Perl has always been known as a useful language, but rarely as one that can help you write fast software. Now, with mod_perl and Apache::DBI, you can write web applications quickly, and watch them run quickly as well.

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.

Letters to the Editor

Various

Issue #52, August 1998

Readers sound off.

Commercial Applications for Linux

In the April “Letters to the Editor” column, two people made comments about commercial applications for Linux. Every time I call a software manufacturer about an application, I ask if they have a version to run on Linux. If not, I ask them to put in a request.

I use Linux as my main operating system at work with Wabi so that I can use MS Office. Now that MS has upgraded Office, I am finding it hard to maintain this plan.

To convince more companies to build software applications for Linux, it might help to run a survey on a web site. The survey would find out how many people are running Linux, if they would be willing to buy software applications, and what price they would be willing to pay. The information garnered from the survey should be made public.

—David Hepner dfh@svl.trw.com

X Display Fix for Lifebook

In response to Michael Scott Shappe, “Accelerated X Laptop Display Server v4.1”, March 1998 (Issue 47), there is a simple fix to the problem of the rightward-shifted X display on the Fujitsu Lifebook (and possibly other laptops with a Phoenix BIOS, where the problem arises).

Go into the BIOS setup, select “Advanced”, then “Video Features”, then “Compensation: Enabled”. The X display will now fill the screen (using either XFree86 or Xi), and you can run Linux in text mode. Windows (I boot both 95 and NT 4.0) will still work with no problems (other than the usual ones).

Some technical details of my laptop: Fujitsu Lifebook 435D, NeoMagic video card and Phoenix NoteBIOS 4.0. It boots Linux (2.0.30), Windows95, NT 4.0 workstation and the NT 4.0 server.

—Dr. Constance A. Stillinger connie.stillinger@kla-tencor.com

Getting Rid of Spam

I enjoyed Mr. Browning's article in the March 1998 issue, “Getting Rid of Spam”, on using **procmail** and the Alcor filters to catch spam. However, I think I've found an easier way to filter spam using procmail. I've observed that spammers seldom send mail to your personal mailing address; instead, they use a mailing list. So instead of seeing a header line like this:

```
To: rsmit06@ibm.net
```

I'm seeing lines like:

```
To: friend@foobar.list
```

Knowing this, it is easy to write a procmail rule to catch it:

```
:0:
*! ^To:.*rsmit06@ibm.net
*! ^Cc:.*rsmit06@ibm.net
/home/rsmit06/.incoming-mail/junk-mail
```

This rule catches virtually 100% of the spam I get. Unfortunately, it also catches the mailing lists to which I subscribe. The solution is to write rules that intercept the valid mailing lists and put them before the “spam-interceptor” in your .procmailrc file. This has two advantages: all mailing lists can be put in a separate folder, and virtually all spam is caught. The only disadvantage is adding each valid mailing list to your .procmailrc file. For me, this isn't a problem.

—Roland Smith, The Netherlands rsmit06@ibm.net

Importance of the GUI

I feel compelled to correct Michael Babcock about what he wrote in his article “The Importance of the GUI in Cross Platform Development” (March 1998) regarding OpenStep. I have been a professional consultant for 14 years and have worked on most workstations, operating systems, languages and engineering paradigms.

I have been staying with OpenStep/NeXT Step/Rhapsody for some time now because it is, in my opinion, the best software development and deployment platform.

OpenStep is not simply a “GUI API (along with some non-GUI functions)...” as he puts it. The very use of the term “API” is misleading. OpenStep is an operating system based on a MACH microkernel, BSD UNIX and object-oriented frameworks, consisting of other frameworks (of objects and classes) for everything from graphics to distributed objects to enterprise computing. This is a radical departure from “APIs” like the ones he discusses in the article; they are not even in the same category.

OpenStep is a complete software solution that lets you write programs using its objects, or extending its objects, and can run on Intel, Motorola, SPARC and HP. Furthermore, the OpenStep OS and development tools run on Windows95/NT, MACH native Intel, MACH Motorola, MACH SPARC, Solaris and HP/UX. With Rhapsody around the corner, we can add Power PC to that list, and most likely Macintosh OS.

As for the complaint about learning ObjectiveC, I have to say that it is much easier to learn than Java and has a smaller linguistic requirement. The hardest thing to learn is that one sends a message to an object with “[object message:arg]” rather than “object.message(arg)”.

OpenStep has a lot of momentum behind it right now. Major Fortune 500 companies and many others have been using it for years. There is no need for “hype” in order to make this technology mainstream. It is a well-established technology that in my opinion could literally save Apple-NeXT. While I am prevented from disclosing details about Rhapsody, suffice it to say this is truly a next-generation technology. It is UNIX, too.

Finally, I wish to say that I applaud the efforts of the GNUStep developers; it would be very useful to have a version that is free, for which source code is available. My hope is that Michael Babcock does indeed get an OpenStep box, so that he can discover his vast underestimation of it for himself.

—Erik Scheirer, sonYx, Inc. boom@sonyx.com

Open Source Editorial

Phil, congratulations on a great magazine. Your editorial in the May issue on Open Source is excellent as far as it goes. Using the phrase “Open Source” in lieu of “free” should overcome many of the negative perceptions and biases that business has with “free”. In addition, “Open” has become a very positive buzzword in IT.

However, there is a crucial element missing in all discussions of Open Source. What is the underlying business model that can allow a business to make money? Though primarily a technology magazine (and a good one), *Linux*

Journal could make a major contribution to the Open Source movement with a short article summarizing the business argument for Open Source. This way, those of us in industry can have a succinct response to “Sounds good, but how do I make money?” Obviously, this short article should stay away from MBA gobbledygook and concentrate on principles. A brief case history would be desirable.

Unfortunately, I am not involved in the Open Source industry and thus can't help with such an article. I can continue to persuade mainstream industry to recognize Open Source as viable software.

—David T. Kjellquist david.t.kjellquist@lmco.com

I agree, this is an issue that needs to be addressed. Russell Nelson has written a guest editorial describing just such a business model. See it on page 10 of this issue.

—Phil Hughes, Publisher info@linuxjournal.com

Summary: Dog eats *LJ*?

Some time ago I wrote here, asking people if their *Linux Journal* magazines were arriving in the mail as dog-eaten as mine. It seems some people get their *LJ* in pristine shape, and this is independent of distance.

Others get theirs in the same shape I do; these people frequently mention that this is puzzling, since they also receive other magazines, which arrive okay. It seems to be only *LJ* that has problems. This, in fact, is true for me.

Thus, I speculate that the size, page-count, binding method or other variables causes *LJ* to be susceptible to mangling in the Post Office machinery. I wonder if *LJ* might consider putting small sticky tabs on some edges of the Journal, to prevent them falling open in Fido Machineries' “Bow-Wow Special Mail Sorter”?

LJ would probably rebel at the suggestion that they send the magazine in an envelope, which, it would seem to me, would be the best preventative measure.

Thanks to all who replied, some with very amusing posts or e-mailed stories.

—Robert Lynch, Berkeley, CA rmlynch@best.com

Actually, we have been trying to find a solution to this problem for some time. Our current printer is not able to polybag domestic mail; foreign mail is outsourced and does go out in an envelope. Our cover is now printed on

heavier paper. When we find an affordable solution that can be handled by the printer, our readers will be the first to know —Editor

Latvian Police Archives

I liked the use of Linux by the Latvian Police force to make a legacy single-user Clipper application available to everyone, although I think the use of DOSEMU is a bit dubious. In the first place, you require a DOS license for each user logged into the system, and second, it is certainly not the most efficient way to run applications. There is a commercial Clipper engine available for Linux (Flagship) which is available for a very modest fee. Using this engine to run legacy Clipper applications instead of DOSEMU should yield better performance with fewer resources and make it easier to evolve the system into something better, perhaps migrating to one of the Linux SQL engines (available both publicly and privately).

—Roger Irwin irwin@trucco.it

Corrections

Archive Index Issue Table of Contents

Advanced search

Open Source Software Model

Russell Nelson

Issue #52, August 1998

Mr. Nelson gives his opinions on how a business can subscribe to the Open Source philosophy and still make money.

Open Source software is all the rage these days. Big companies and small companies alike are either releasing their software as Open Source or considering it. On the forefront of their minds is “Yes, it's more reliable; yes, it meets more users' needs. But how do we make money on it?”

First, we must be clear that no industry is sacrosanct in any economy. Almost all companies rely on special protection from the government, called intellectual property rights. These rights, covering both copyright and patent laws, exist because they are thought to be economically efficient—that is, they create more than they destroy. Should that prove wrong—that protecting computer software as intellectual property actually makes society poorer—then the computer industry will just have to rely solely on contract law. If society is better off with all software as Open Source, then that is how it will be, even if it makes computer programmers worse off financially.

Open Source software need not impoverish all programmers—indeed, there is good evidence that the deleterious effects of switching to all Open Source software would be only short term. Remember the Luddites? The dislocations in the weaving industry were only temporary. Forty years later, the industry was employing just as many weavers and, of course, weaving a greater amount of cloth.

If you want to see this model in action, look through this issue of *Linux Journal*. You'll see ad after ad, most of them for Open Source software. The way to make money in a free market is to solve someone's problem and take a cut of the benefit they receive. There are many different ways to do so, as reading the ads proves.

Second, I have been assuming that Open Source software is applicable to all types of software. This is not clearly true or false. What is definitely true is that an operating system can be Open Source and be successful, as all the readers of this magazine know. Perhaps other types of software are not as amenable to Open Source distribution. If this is the case, then the creators of such Open Source software will either not exist in the first place or will go out of business. For this type of software, the proprietary model would work best.

People have gotten rich from Open Source software, and still are. As long as you have an industry where people can get rich, you've got plenty of incentive to enter the industry.

The big question being discussed on the Free Software Business mailing list (fsb-subscribe@crynwr.com) is: “How do you create a piece of software which requires a large up-front investment of human energy?”

Companies such as Netscape or Corel can easily afford to free a major piece of code. The software has been proprietary for a number of years and has recouped its initial investment. The Crynwr Packet Driver Collection started small and was initially useful with only one driver in it, so it doesn't fit the open model. The first released version of Linux was a real UNIX kernel with a file system and serial port drivers. To the extent that it needed an up-front investment, Linus has been paid back for it.

No, recovering the investment is a tough job without intellectual property laws. Problem is, they're needed precisely for only those tough jobs. More than any other type of intellectual property, creation of most computer software involves very little capital costs. Invoking the coercion of the state (even as far as stealing a person's thoughts, when independent creation occurs) is less and less appropriate. Software creation is cheaper now than when PCs were new. The areas where Open Software cannot succeed seem smaller and smaller.

Russell Nelson has been developing Open Source software long before the term existed. His first freed software, Freemacs, came out 15 years ago. He was the progenitor of the Clarkson Packet Driver Collection, the first freed software to be recognized by *PC Magazine*'s Technical Excellence awards. Currently, he works for Crynwr Software, founded in 1991, an Open Source support firm. He can be reached at nelson-lj@crynwr.com.

Archive Index Issue Table of Contents

Advanced search

Sun Joins Linux International

Marjorie Richardson

Issue #52, August 1998

In a year when Netscape has released their source and many companies have announced that their products will be supporting Linux, I felt Sun's move was an interesting enough development to want to know more.

In May, Sun Microsystems joined Linux International. In a year when Netscape has released their source and many companies have announced that their products will be supporting Linux, I felt Sun's move was an interesting enough development to want to know more. Therefore, I did a short e-mail interview with Charles Andres, a Group Manager in Market Development Engineering at Sun Microsystems. Here's what he told me.

Margie: Why has Sun made the decision to join Linux International?

Charles: Sun Microsystems is responding to renewed interest in running Linux on its UltraSPARC products, such as the Ultra 5. SPARC products have always been designed to run UNIX extremely well. Linux runs well on UltraSPARC platforms.

It is important to note that this move in no way diminishes Sun's support for Solaris, a proven reliable scalable operating system. The Solaris environment will still be provided with all SPARC systems and is considered by us to be the best operating system for enterprise and network computing.

Margie: Is Sun planning to have Linux support for all its products?

Charles: Sun Microsystems is not planning on selling any products that are bundled with Linux. Sun bundles Solaris with every workstation and server it currently ships. There are also no plans to provide support for Linux directly. However, there are a number of Linux vendors that support a variety of platforms. We are working to ensure that these vendors include UltraSPARC platform support for their Linux products.

Margie: Does this move represent a shift in policy for Sun? Last year, we asked for a picture of a SunSPARC workstation to use on our cover, and were refused because “Linux is a competitor.” (We used a Ross SPARCplug instead.)

Charles: Sun Microsystems has never had an official policy regarding Linux up to now. As stated above, Sun Microsystems has gone from having no policy regarding Linux, to helping to ensure that Linux runs on SPARC by assisting companies who sell supported versions of Linux.

Margie: How does Sun feel about the “Open Source” movement? (Prominent in the news, because of Netscape source release.)

Charles: Sun Microsystems has a long tradition of supporting open standards, typically through standardized interfaces, many of which Sun has invented. Providing source code may be appropriate in some specific instances, but typically works well only in situations where trademarks associated with the source code are licensed. Compatibility, consistency, reliability and upgrades require a business model that can finance the effort required to provide them.

Users who want the freedom of Open Source take on the responsibility of maintaining their own source code, but cannot guarantee consistent results with other variants. This could become a problem for Netscape source variants if they are not uniquely identified. This is why we feel brand protection through licensing is so important.

Margie: Some people feel that Java should be made Open Source. Any chance of that happening?

Charles: Source for the Java language is available to anyone who signs the Java license which is free for non-commercial use. This is done to allow Java to run anywhere, and to avoid problems that could occur when source is modified to produce variants that are not consistent with the Java language specification.

Margie: Anything else you'd like to add?

Charles: We look forward to working with you and the Linux community to promote the advantages of UNIX and Linux on SPARC in the future.

Margie:Thank you for your time.

News Flash!

Archive Index Issue Table of Contents

Advanced search

Linux Directory Trees

Matus Telgarsky

Issue #52, August 1998

A quick tour of the various directories in Linux and the files contained in each.

Important to solving many problems in Linux is understanding where files are located in the directory structure. There is, however, no “absolute” or “perfect” directory structure; since Linux is so configurable, the same files can be put in literally any place and everything will still work. The home directory for the user joebob will customarily be /home/joebob, but can be set to be /.joebob. In order to prevent everyone from being constantly confused, however, a general standard was formed.

Unlike some other operating systems, Linux does not have all of its files in one enormous, messy directory. The files are distributed neatly, and once you have a basic understanding, it is easy to find and modify any file you wish. This article will go through the directory tree in alphabetical order by directory name with references to other directories.

The first directory is /bin which is short for binary, the file format of most executables. Basic executables for commands, such as **ls**, **sync**, **rm** and **cat**, are contained in /bin. Other directories containing binary files are /usr/bin/, /usr/X11R6/bin, /usr/local/bin and ~/bin.

The next directory is /boot. Not all distributions have this directory. If present, it usually contains kernels and boot maps for booting. I keep kernels in the / directory.

Next is /cdrom, where the CD-ROM drive is mounted in some distributions. Others may mount it in /mnt/cdrom.

Next is the /dev directory, which contains many files (**ls** shows 1041 on my machine). These files are device files (and sometimes FIFOs and other files, such as the text /dev/sndstat) which allow you to access devices. These are not

actual drivers; they are just interfaces to the devices. Important files such as /dev/mixer and /dev/fd0 are stored here.

Next is the directory /etc, a very important directory indeed. It contains data files for many programs, including the configuration files for LILO. The kernel may save sound device configuration to /etc/soundconf. Scripts run when Linux starts up are usually kept in /etc/rc.d. The way files are set up in rc.d changes in different distributions. Another important file is /etc/inittab which contains some system settings, such as run levels and what happens when **ctrl+alt+del** is pressed. To keep anyone from rebooting my machine, I set **ctrl+alt+del** to run **amp** and play an mp3 file. Many servers have commented out that line to prevent the machine from being rebooted by unauthorized users. Another important place is /etc/X11, which can contain your XF86Config file; however, this file may be in many other locations. Also in this directory is the subdirectory /etc/xdm, in which you can customize your XDM login for some amazing looks. (Refer to http://torment.ntr.net/xdm/ for more information.)

Next is the directory /home. This directory contains home directories for users. The subdirectories /home/ftp and home/http may be here for use by their respective daemons connect.

The directory /lib, like /bin, contains basic executables and the basic libraries. The directories /usr/X11R6/lib, /usr/lib and /usr/local/lib are other repositories of libraries.

Sometimes you will see the directory /lost+found. This directory may contain inodes which were somehow “lost” in the file system with no other location in which to be placed. An *inode* is the data structure on disk in which a description of a file's attributes is stored.

Some block devices, such as floppy and CD-ROM drives, are mounted in the /mnt directory. Some distributions use /cdrom for CD-ROMs and /mnt for floppies, while still others use /mnt/cdrom for CD-ROMs and /mnt/floppy for floppies.

Next is /proc, quite a curious directory. Notice that as root you cannot edit any of its files. (If you can, it is a bad thing.) This directory contains kernel data and has its own file system (/proc) and strangely enough, typing **du** in /proc returns the value 0. The kernel stores all sorts of data in this directory, which is why you shouldn't edit it.

The /root directory is the home of root. Often, in /root, you will find programs or scripts written by root that perform some task for the machine.

The /sbin directory contains binaries that normal users generally don't need—binaries meant for superusers. Other sbin directories are /usr/sbin, /usr/local/sbin, etc., which contain **ldconfig**, **mke2fs**, **quotaon** and other programs not needed by regular users.

The /tmp directory is unique in that it has write permissions set for everyone. This directory contains temporary files and FIFOs for many programs.

Now we come to /usr, probably the largest directory on your computer. /usr contains many files for users (as the name implies) and their programs. It has subdirectories similar to those in the root directory. The directory /usr/X11R6 contains files for X. Here you will find man pages, executables, libraries and include files for X. (As a side note, the X include files in /usr/include are linked to /usr/X11R6/include/X11.) In /usr/bin there are many more executables such as **passwd**, **nice** and **zgrep**. The directory /usr/doc may hold help files for the system. The /usr/lib directory contains more libraries, and directories for different programs that don't contain libraries for those programs but rather data files. The next subdirectory is the all-powerful /usr/local. If you compile and install a program, it will most likely be installed in /usr/local. Those who did not upgrade distributions and wanted to install glibc by hand (me), found that glibc was installed in /usr/local/include instead of /usr/include as it is on newly installed systems. Many programs, including our favorite the GIMP, will install some data files in /usr/local/share. Since it contains programs you install, /usr/local will probably become your favorite directory. /usr/src is where the kernel source should be kept.

Finally, there is the /var directory in which you will find e-mail spools, printer spools, files containing scores for games and nothing else of much interest.

Clearly, the directory tree in Linux isn't all that complex. One of the best features of Linux is that if things go wrong, you can fix them. Rebooting and reinstalling your OS isn't your only option. To tell the truth, I have never had to do so. With an understanding of where files are located in Linux, system maintenance becomes easier.

Matus Telgarsky is a high school student who has been happily running Linux and using it for all computer-related tasks for many years now. He wishes to give everyone the experience of Linux, enjoys playing violin and waits for Enlightenment .14 to come out. Send him e-mail at matusa@nmsu.edu.

Archive Index Issue Table of Contents

Advanced search

Implementing a deltree Command in Linux

Ph.D.. Graydon L. Ekdahl,

Issue #52, August 1998

Removing a software package is made easy by using Dr. Ekdahl's deltree command.

Ever needed to excise a large software package from your file space only to discover it dispersed over a directory tree containing over a hundred files and one tenth as many subdirectories? The command **rm -rf** will clear everything away nicely. However, in order to learn more about walking a Linux directory tree, let's look at implementing **rm -r** as a home-brew (DOS-like) **deltree** command in Linux is not difficult and will make it easier to remove unused or unwanted software packages if you own the utility and use it in your own file space.

Linux C Library Resources: Files and Directories

To select C library resources to complete this task, we first determine which resources are generally available in UNIX and then which of these resources Linux implements. Two UNIX functions contained in the header file ftw.h walk a directory tree:

```
int ftw ( const char * path,
        int (*funcptr )
        ( const char *, const struct stat*, int ),
        int depth )
```

and

```
int nftw (const char * path,
        int (*funcptr)
        (const char *, const struct stat*, int, struct FTW*),
        int depth,
        int flag )
```

Linux does not implement the second function, so we turn to the first; **ftw** walks a directory tree from top to bottom. For each directory entry, ftw calls the function pointed to by **funcptr** with the name of the entry, a pointer to a **stat**

structure containing inode information and a flag set to convey information about the directory entry in question:

- **FTW_F**: a file
- **FTW_D**: a directory
- **FTW_DNR**: a non-readable directory
- **FTW_NS**: stat failed and inode information is not available.

What does ftw return to the caller? If it completes a successful walk through the tree, it returns **0**. Otherwise, it returns **-1** and sets the global error flag **errno** appropriately. Using ftw, it is a simple matter to create a function designed to delete directory items and to pass a pointer to that function to ftw (see Listing 1):

In **DelEntry()**, the C library function

```
int remove ( const char * path )
```

in stdio.h does the actual work. This function returns **0** if successful, **-1** if unsuccessful and sets the global variable **errno** as necessary to handle a number of different error conditions, which the Linux man pages explain in detail.

There is, however, a catch. In UNIX, **remove** may generally be used to delete either files or empty directories. In Linux, **remove** only processes files and would therefore empty the directory tree but leave the tree itself standing. Two UNIX C library functions may provide solutions:

```
int rmdirp ( char * d, char *d1 )
int rmdir ( const char * path )
```

Linux does not implement the first UNIX function **rmdirp**, so we focus on the second. The function **rmdir** removes only empty directories and returns 0 on success, **-1** otherwise with **errno** set. To accomplish our task, we must walk the tree twice: once from the top down to the directory at the bottom, deleting files as we go, and the second time from the bottom back to the top, removing empty directories in reverse order. The perfect tool to achieve this result is a container class: a stack of pointers to directory path names.

StrStack: A Stack of Pointers to char Arrays

When ftw calls DelEntry, it supplies a flag indicating whether it found a file, a directory or cannot cope, and we can use this flag to fill **StrStack** with path names inside DelEntry as ftw walks the tree. The question is where to put the stack. The header file ftw.h specifies the signatures of the ftw function and the function pointed to by **funcptr**, and neither signature includes a stack, so we

cannot pass the stack in by reference as a parameter to DelEntry. The simplest solution is to create **StrStack** as an external variable in the implementation file funcs.cpp which holds the function definitions for **main**. As an external variable, **StrStack** will be equally accessible to DelEntry and to DelDirectories, provided it is defined in the implementation file above these functions.

Several aspects of StrStack require explanation. StrStack differs from the average stack in that each node contains pointers to two different, dynamically allocated structures: a pointer to the next node and a pointer to character strings of varying lengths. Two allocations are necessary to create a node, and two separate deallocations are necessary to destroy a node. By making StrStack responsible for both allocations, the code is more reliable, more robust and has no memory leaks. In addition, if the caller was responsible for allocating and deallocating memory containing character arrays, then exocode could literally pull data out from under StrStack, leaving dangling pointers.

In some places, implicit recursion accomplishes allocation and deallocation, and it may not be obvious at first glance how the process works. Let's examine the copy constructor for StrStack shown in Listing 2.

The class copy constructor is designed to create a copy of a StrStack object in case a function ever passes a stack in or out by value. The code in the function is obvious except for one line:

```
next_ = new strNode ( *srcnode.next_ );
                // indirect recursion
```

This line is an example of indirect recursion, and it duplicates all the nodes in the StrStack node sequence. How does it work? The argument to **new** is: **strNode (*srcnode.next_)** which is another call to the node copy constructor with the next node in sequence as argument. As long as each node contains a pointer to another node, the copy constructor repeatedly calls itself recursively until it encounters a **NULL** in the **next_** field of the last node in the sequence. With that, the recursion ceases and begins to unwind, constructing a copy backwards from the tail of the node sequence to the head. Note that the copy constructor deals, as promised, with two different dynamic allocations: allocating memory for the node, and then for the character array which holds the path name. In the node destructor, the line **delete next_ again** triggers a sequence of implicit recursions which result in the destructor calling itself until the final **NULL** at the end of the list is encountered. At that point, the recursion unwinds, and nodes are deleted from the tail of the list back to the head.

More Utilities

If C library calls do most of the grunt work, writing a simple utility like deltree is neither difficult nor time consuming and may offer unanticipated opportunities to use the framework of the program to address additional problems. This code can be adapted to perform the same functions as **find**; use it as a skeleton for a **findfile** command which scans a directory tree for file names matching a command-line argument. Just replace the file and directory deletion subroutines with functions that compare each file name with the target name and print out the path name to each match.

Do you port code from the Borland or Watcom PC DOS or OS/2 environments and move entire directory trees into Linux space at once? If so, then you have probably discovered that unwanted files migrate along with the necessary ones: files with suffixes such as .map, .sym, .dsk, .swp, .prj, .exe, and the like. With modification, deltree can also provide a framework for a **cleandir** utility that removes the chaff from the toolbox directory tree. To make the necessary changes, replace StrStack with a StrList class which contains a list of target file name suffixes. Instead of removing all files, the utility checks the suffix of each file in the directory tree against the list of target suffixes and deletes selectively. Once you have the hang of walking a Linux directory tree, creating plug-in functions to perform other tasks is a simple matter, and it is easy to generate a group of utilities which address a broader spectrum of directory tree maintenance issues.

All code needed to implement this command is available by anonymous download in the file ftp://ftp.linuxjournal.com/pub/lj/listings/issue52/2439.tgz.

Graydon Ekdahl (gekdahl@ibm.net) s president of Econometrics, Inc. located in Chapel Hill, N.C. Graydon enjoys creating database applications and is interested in data structures, algorithms, C++ and Java.

Archive Index Issue Table of Contents

Advanced search

Linux Means Business

John Taves

Issue #52, August 1998

Mr. Taves tells us how he set up the USPS with Linux boxes for running OCR software and image capture.

The United States Postal Service deployed over 900 Linux-based systems throughout the United States in 1997 to automatically recognize the destination addresses on mail pieces. Each system consists of five dual Pentium Pro 200MHz (PP200) computers and one single PP200, all running Linux.

The USPS already had the mail-piece scanners and some old custom computers that recognized the addresses. This project connected the Linux computers to each scanner in order to run more modern OCR (optical character recognition) algorithms supplied by RAF Technology Inc. This system was designed by me during the time I was Vice President of Engineering at RAF Technology Inc.

One of the five Linux boxes has a monitor, keyboard, mouse, CD-ROM and floppy—the other four are headless. Each has 128MB RAM and a 2.5GB hard drive. The mail pieces are scanned at 212dpi at a rate of 12 per second. The binary image is sent to one of the Linux boxes via a custom cable and receiver board. The board packs the bits and uses DMA (direct memory access) to transfer the data over the PCI bus. The receiving computer runs a process that compresses the images and routes them via Ethernet to one of 10 identical processes, two for each CPU, that do the hand-print recognition and machine-print recognition. Those algorithms recognize the text from the image in less than a second and return the ASCII results to a database on a separate computer that looks up the zip code. The slave computers are connected on a subnet with the master which has a second Ethernet card connected to the rest of the computers associated with the scanner. The local network is 10Mbps Ethernet and handles the compressed binary images sent to the slaves and the ASCII results received from the slaves.

Originally, I planned to run only machine-print recognition algorithms, which use far fewer resources than hand-print recognition algorithms. The majority of the mail stream is machine-printed addresses. The original system was designed to have four single PP200s, each with 32MB RAM. Only one was to have a CD, hard disk and monitor. The others, called slaves, would have no mass storage and would boot off the ROM on the Ethernet card. Just before the system was to be deployed, the USPS decided that the hand-print recognition developed by SUNY Buffalo was accurate enough to include in the system. We reconfigured the slaves to have a 2.5GB hard disk and 128MB RAM.

I shopped for the best combination of size, speed and cost for the slave computers and settled on a tidy 3.5-inch high rack chassis from Tri Map International. One other choice was a passive back plane with up to four PC-on-a-card computers. PC-on-a-card computers are basically ISA cards with a complete PC on them. The card gets power from the back plane, and you can plug four cards at a time into one back plane, since the back plane is separated into four separate ISA busses. The theory is that you can put more CPUs into a smaller package. The passive back plane takes up 7 inches of vertical space on the rack for the four CPUs. The width and depth of the rack is basically fixed. The problem was that PC-on-a-card computers were quite expensive and didn't support the fastest CPUs. At that time, I could only get P166s on a card when PP200s were available in the Baby AT form factor. The Baby AT form factor is the extremely popular PC motherboard type that has the big round keyboard plug. I was much more comfortable recommending the baby AT form factor because replacement motherboards would be available forever. The best choice turned out to be the nifty 3.5-inch high chassis. So for 7 inches of vertical space, I got two PP200s instead of four P166s, at a much lower cost and with a more common motherboard.

As I worked on this project, computers just kept getting faster. By the time we deployed, we put two dual PP200s in the 3.5-inch chassis and the PC-on-a-card computers were still only capable of taking P200s. It is not clear what would be the best choice today. Pentium IIs come on a big riser card that won't fit in the 3.5-inch chassis. The 3.5-inch chassis required a special board that bent the Ethernet card 90 degrees, so it was parallel to the motherboard. This limited the number of cards that could be plugged in; fortunately, I needed only the Ethernet card.

Linux was an excellent OS for this application. To make the OS and OCR software run in 32MB RAM with no swap, the kernel was recompiled with only the essentials—an impossibility with a Microsoft OS. Because Linux is free, I didn't have to worry about license fees. The device driver for the custom card was relatively painless to develop, and I must say Linux “product support” was far superior to anything else I've used. When I had trouble allocating large

amounts of real memory in the kernel, I e-mailed a question and always received a quick response. As it turned out, I needed to muck with the source code and recompile, as a large amount of real memory was needed for a side project in which the USPS wanted to capture 8-bit gray-scale images.

The scanners, already in place, could produce 8-bit gray-scale data. Unfortunately, at 12 images per second the gray-scale data arrives at a rate of 28MBps. The capture card and PCI bus would be able to deliver the data to the PC's RAM, but the disk wouldn't be able to store it fast enough. I set up the PCI card to fill RAM with images until no space was left. The data would then be read from the device driver and written to the disk. While the data is written to the disk, the scanner continues to run at 12 mail-pieces per second, but the PCI card just ignores the images until the RAM has been freed up. With this system, the USPS can capture a decent sampling of gray-scale mail-piece images in real time.

The USPS told me they were having another company build a custom, portable, gray-scale, data-capture machine, in order to be able to capture sample images throughout the USA. I told them an expensive custom solution was not necessary because I could build one in a few weeks for a few thousand dollars each. I did this by putting together several luggable Linux boxes. I used the portable-style computer case that has a color flat-panel display and a full-size keyboard that snaps against the screen. Since I needed to put a tape drive and a PCI card in the PC, I couldn't use a laptop. I had no problems getting an X Window System server running on the portables. The setup would have worked without X, but it was important to the USPS to be able to look at the images on the spot. These data-capture units, as setup, gave them just what they needed and worked great.

John Taves lives in Redmond Washington with his wife and 2.5-year-old boy. He has done OCR software development for 13 years and built a very profitable OCR business. He is currently looking for employment as a General Manager, CTO or Development Manager of a software company or business unit. He is also interested in doing contract development. More importantly, his wife is the second best woman croquet player in the world, and he is currently the top ranked croquet player in the United States. They expect their son to play a real sport. John can be reached at jtaves@aa.net.

Archive Index Issue Table of Contents

Advanced search

New Products

Amy Kukuk

Issue #52, August 1998

ION 1.1, Projector 1.00, Swift View and more.

ION 1.1

Research Systems, Inc. has announced the release of ION (IDL on the Net) 1.1. ION, which is built on IDL (Interactive Data Language), is designed for high-performance data transfers and implements a client-server architecture. All information is passed between the server and client in a binary, platform-independent network format. ION will be sold as an unlimited-connection license, meaning any number of web-based clients can connect to the ION server at the same time. An introductory price of $6,995 US is valid until December 31, 1998.

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

Projector 1.00

The release of Projector 1.00 has been announced. Projector 1.00 is the initial release of a scheduling tool that accepts a list of project sub-tasks with their interdependencies and generates Gantt charts and schedule analysis reports. Projector 1.00 reads a project definition file, containing information about the activities in the project, and produces a two-part analysis of the schedule. Projector 1.00 is available on diskette for $200 US per system.

Contact: Morris Dovey, 1001 Office Park Road, Suite 300, West Des Moines, IA 50265, E-mail: mrdovey@iedu.com, URL: http://www.iedu.com/project/.

*Swift*View

Northern Development Group, Inc. has announced the availability of *Swift*View Plug-in for UNIX-based web browsers. *Swift*View provides instant "viewing" of technical and business documents and drawings on the Internet and corporate Intranets. This product interactively displays output that would normally be printed out on HP LaserJet printers and plotters, using their native PCL and HPGL output file formats. The license enables an unlimited number of Intranet or Internet users to access documents or drawings from a licensed web site for $1500 US.

Contact: Northern Development Group, Inc., 7100 SW Hampton, Suite 207, Portland, OR 97223, Phone: 503-620-0196, Fax: 503-639-8466, E-mail: sales@ndg.com, URL: http://www.ndg.com/.

WebSite Professional 2.2

O'Reilly & Associates has announced the release of WebSite Professional 2.2. WebSite Professional version 2.2 includes Uplink, a utility designed for Internet Content Providers (ICPs) and Internet Service Providers (ISPs). Other new features include enhanced log file management and generation, the inclusion of Live Software's new JRun 2.1, and support for Java Development Kit (JDK) version 1.2 of the JavaSoft Servlet Advanced Programming Interface (API) 1.1. Suggested list price for WebSite Professional 2.0 is $799 US. The upgrade to version 2.2 is free for downloading by registered version 2.0 and 2.1 customers.

Contact: O'Reilly & Associates, 101 Morris Street, Sebastopol, CA 95472, Phone: 707-829-0515, E-mail: software@oreilly.com, URL: http://software.oreilly.com/.

FlashCONNECT

Pick Systems, Inc. has announced FlashCONNECT, a new tool available for use with D3, the Company's DBMS. FlashCONNECT enables application developers

familiar with Pick Systems' FlashBASIC programming language to quickly build dynamic, database-driven web sites and web-enabled Intranet databases without requiring extensive modifications to existing applications. FlashCONNECT allows the creation of dynamic web page components that incorporate a consistent look and feel. FlashCONNECT is activated through a one-time fee of $1000 US per database server.

Contact: Pick Systems, 1691 Browning, Irvine, CA 92606, Phone: 717-261-7425, Fax: 714-250-8187, E-mail: sales@picksys.com, URL: http://www.picksys.com/.

iScript

Servertec has announced iScript, a platform independent scripting language written in Java, for creating scalable, server-side, object-oriented, n-Tier enterprise solutions. The iScript scripting language features platform independence, object-oriented architecture, web-server integration, support for Common Gateway Interface (CGI), dynamic content generator, static content preprocessor, make facility, just-in-time pcode generator and caching, Java API wrappers and open component API. Servertec will be releasing the iScript Developer Kit, which integrates the iScript scripting language, documentation and examples to the public. Web site developers can use this kit to create and maintain dynamic, data driven and static web sites. iScript preview release is available for free at http://www.servertec.com/. The final release of iScript is scheduled to ship in the second half of 1998. Final pricing details have yet to be announced.

Contact: Servertec, 18 Oakwood Avenue, Kearny, NJ 07032, URL: http://www.servertec.com/.

S.u.S.E. 5.2

S.u.S.E., Inc. has announced the latest release of its flagship product, S.u.S.E. Linux version 5.2. This version includes enhancements to YaST, S.u.S.E.'s menu-driven system management tool, which guides users through installation and setup. In version 5.2, YaST enables the configuration of PPP Internet access and scanner support, and for the first time includes the full source code. S.u.S.E. Linux 5.2 ships with over 800 Linux software packages. Included among these is the Netscape Communicator 4.04 web browser and KDE Desktop b3. The price for the new version of S.u.S.E is $49.95 US.

Contact: S.u.S.E., Inc., 458 Santa Clara Avenue, Oakland, CA 94610, Phone: 888-875-4689, E-mail: info@suse.com, URL: http://www.suse.com/Products/lx52/index.html.

Archive Index Issue Table of Contents

Advanced search

Linux as a Backup E-mail Server

John Blair

Issue #52, August 1998

Implementing a fall-back e-mail server is easy to do by setting the proper entries in the DNS server and running sendmail on a Debian Linux system.

One Friday morning several months ago, the Microsoft Exchange e-mail server I'm in charge of crashed. At the time, I was a fairly new Windows NT administrator. The server, which we call tuccster, was not running properly again until late the following Sunday evening.

Unfortunately, at this point my problems had just begun. Our departmental e-mail server had been down for nearly three days and megabytes of important e-mail were spooled on remote servers all over the Internet. There was no way to predict when the mail would arrive and how much e-mail might never show up. Since different e-mail servers try to re-send e-mail at different intervals, what did show up would arrive out of chronological order. While I had learned a lot about Windows NT disaster recovery, the whole event was a major inconvenience for the users in my department and a horrible embarrassment for me.

While I'm not convinced I could have prevented the failure of Microsoft Exchange, I could have set up a fall-back e-mail server to spool all incoming mail while tuccster was down. Shortly after the mishap, I was able to find an old Gateway 486/66 with an Ethernet card that was being replaced with a faster Pentium system. Using Debian Linux and sendmail, I set up a fall-back e-mail server that receives and spools any incoming e-mail whenever tuccster, the primary Windows-based server, is down. The addition of a web server and a simple CGI script written in Perl provided a simple user interface into the system. I configured the web server so that a web browser could be used from certain trusted hosts to check if anything is waiting in the sendmail queue. Once our primary server is ready to begin receiving e-mail, the sendmail queue can be flushed by clicking a link on the same web page.

DNS MX Records

A “fall-back e-mail server” is an old idea on the Internet—the functionality to set one up is actually built into the Domain Name Server (DNS) protocol. The intent was that every important e-mail server would have a backup in place. A Domain Name Server contains many types of records. The most common of these types are SOA records which indicate authority for a domain's data; NS records which list name servers for a domain; *A* records which map a name to an address; PTR records which perform the reverse, mapping an address to a name; and MX records which describe “mail exchangers”. MX records allow one to define the actual host responsible for receiving mail directed at any particular host. The host actually responsible for receiving e-mail need not be the host to which the mail appears to be addressed.

To illustrate why this would be useful, imagine a set of workstations called larry, curly and moe. To reduce the load on curly and moe, we would like all incoming e-mail to be directed to larry, regardless of the host to which the mail was actually addressed. MX records provide a way to achieve this goal. Suppose we program our DNS server with the following:

```
larry.tucc.uab.edu.  IN  MX  1  larry.tucc.uab.edu.
curly.tucc.uab.edu.  IN  MX  1  larry.tucc.uab.edu.
moe.tucc.uab.edu.    IN  MX  1  larry.tucc.uab.edu.
```

If somebody tries to send e-mail to foo@moe.tucc.uab.edu, the mail transport agent (MTA) should look up the DNS record and see that larry is responsible for all e-mail directed to moe. Not all MTAs properly implement MX redirection. The mail will then be delivered to larry as if it were addressed to foo@larry.tucc.uab.edu.

While this is useful, it is not all that can be accomplished with MX records. The number appearing in the example between “MX” and “larry.tucc.uab.edu” is a preference value. Suppose I was worried that student projects running on larry might cause the system to crash periodically, or that larry was running a less-than-robust e-mail server. I could set up curly as a fall-back server by using the following DNS entries:

```
larry.tucc.uab.edu.  IN  MX  1  larry.tucc.uab.edu.
larry.tucc.uab.edu.  IN  MX  2  curly.tucc.uab.edu.
curly.tucc.uab.edu.  IN  MX  1  larry.tucc.uab.edu.
curly.tucc.uab.edu.  IN  MX  2  curly.tucc.uab.edu.
moe.tucc.uab.edu.    IN  MX  1  larry.tucc.uab.edu.
moe.tucc.uab.edu.    IN  MX  2  curly.tucc.uab.edu.
```

Now suppose that larry is down for some reason. A remote host attempting to send e-mail to larry would discover that larry is unavailable. It would then learn from DNS that curly is the next preferred e-mail server for larry. The remote host will send the message to curly. The mail transport agent (such as

sendmail) on curly will then realize that larry is preferred over curly as a mail exchange. It then spools the message locally, periodically attempting to pass the message on until it succeeds.

Setting up a Fall-back Server Using Linux

You've probably guessed by now that I used this very same technique to set up my fall-back e-mail server. Serendipitously, one of the older 486 computers in the department was slated for replacement with a shiny new Pentium-based PC. This computer, with 16MB of RAM, a 600MB hard drive and an Ethernet card was the perfect computer for the job. Setting up the computer was relatively easy. I chose to install the Debian distribution, since I'm most familiar with it. I could just as easily have used any of the other distributions. The components essential to my setup are the latest version of sendmail, the Apache web server (although NCSA, CERN or another web server would have worked) and Perl. I also installed other tools, such as Emacs, gcc, make, rcs and vi to make myself more comfortable while working on the system.

These utilities also allowed me to rebuild the kernel without using another system, and they should come in handy if I ever have to repair the system after some sort of mishap. I left all X11 libraries off the system to save disk space. After compiling a kernel containing just the modules I needed to run the system, I created a rescue boot floppy with this same kernel. There's no reason not to be prepared for the worst. Finally, I dubbed the system bartleby, after the downtrodden scrivener in the Dead Letter Office of Herman Melville's short story *Bartleby the Scrivener*.

The next step was to choose an appropriate location for bartleby to live. Simply placing bartleby in the same room as tuccster would provide adequate backup if tuccster crashed of its own accord again. However, by now I had disaster recovery on the brain and wanted to protect against other sorts of failures. I finally settled on a location in our mainframe room, which is located in a different building. This placed bartleby on a different subnet than tuccster, so no e-mail would be lost if our own subnet failed. Being in the mainframe room also meant that bartleby was located on a protected power system.

Long after tuccster has been automatically shut down by its UPS, bartleby will be merrily spooling any incoming e-mail. The wisdom of placing the system on a separate subnet was proven several months later when our own subnet was accidently disconnected by maintenance workers. With bartleby properly in place, all that was left was adding the appropriate entry to our DNS and configuring sendmail and the web server.

Just as in my example with the Three Stooges, I wanted to add MX records to our domain name server so that both bartleby and tuccster were listed as MX

hosts for tuccster, with tuccster as the preferred host. The two MX records as I entered them are shown in Listing 1.

While configuring sendmail is often something to fear, it was easy in this case. **sendmail** will correctly process deferred e-mail by default. All that was necessary was configuring sendmail to send and receive e-mail directed to and from bartleby, something handled automatically by the Debian package install script. Once I finished the sendmail install, bartleby was working correctly as a fall-back e-mail server. The next time I brought tuccster down for maintenance, I examined the sendmail queue on bartleby using the **sendmail -bp** command. Sure enough, several e-mail messages were in the queue, waiting for tuccster to wake up again. After I brought tuccster back on-line, I flushed the queue using the **sendmail -q** command.

It is possible to adjust the frequency with which sendmail attempts to process any spooled messages by appending a time argument to the **-q** command when sendmail is first started. The Debian package for sendmail includes a macro at the beginning of the sendmail startup script that makes setting this value easy. I set mine to 30 minutes (the actual command-line argument is **-q30m**). The sendmail man page describes the syntax for the **-q** command. I automatically shut down the Exchange server late each Friday night to save its message store to tape. I wanted bartleby to flush its queue automatically so any messages that were deferred during the backup would arrive properly without human intervention. You may wish to lengthen the time between queue flushes.

Creating a Simple Web-based Interface

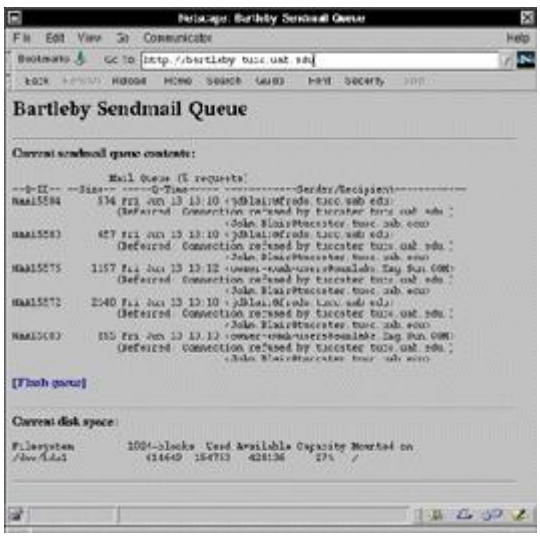
The system as I had it set up was pretty good. However, it did require at least a minimal understanding of the UNIX shell to log in and check or flush the sendmail queue. I wanted to make the system as foolproof as possible in the event something happened while I was away from the office. I settled on a simple CGI script written in Perl. This script (see Listing 2) displays the current contents of the sendmail queue, as well as the output from the **df** command. This allows someone to see at a glance what e-mail messages are spooled in the system and how much disk space remains. If the sendmail queue is not empty, a link is created that can be clicked to flush the queue.

Since this is the only service this web server provides, I called it “index.cgi” and placed it in the root of the server directory. Adding “index.cgi” to the DirectoryIndex list in the srm.conf configuration file causes the script to be executed automatically when the server is accessed. I also adjusted the MinSpareServers and MaxSpareServers values in the httpd.conf file, so that no spare servers are left running. In this case, I don't mind slowing the response time to free up a little extra memory. There were a few final security issues to

contemplate. I consider the listing of the sendmail queue to be “sensitive but unclassified” information. By this, I mean I don't feel an intruder would find the information contained therein particularly useful, but then again I don't see the need to make the information easily available to the entire Internet. The action of flushing the queue is nearly harmless on its own, though I suppose it would be possible to mount a denial-of-service attack by repeatedly flushing the queue. In the end, I decided to allow non-password based access from a handful of machines in the department.

Each of these machines is located in the office of an administrator who might have a reason to check the queue. I implemented these restrictions with the Apache directives shown in Listing 3 in the access.conf file.

I could have added the standard plaintext password authentication available through nearly every web browser, but I don't believe it would have added enough real security to offset the inconvenience. A screen shot of this web interface is shown below.



I decided to restrict shell access to myself. Furthermore, I installed the s/key system from Bell Labs to avoid plaintext passwords and used TCP Wrappers to restrict the hosts that can log into the system. While there is little to be gained (other than, perhaps, traffic analysis) from examining the sendmail queue, much could be gained from unrestricted access to any incoming e-mail message.

Conclusions

Setting up a fall-back e-mail server using a Linux system running on older hardware is an excellent tool to preserve incoming e-mail in the event of a disaster on your primary server. While I am backing up a Microsoft Exchange server, the same technique can be used to back up an SMTP server from any vendor. Setting up the fall-back server costs nearly nothing other than the time

required for configuration. Having a very stable system completely independent of the rest of our network has also proven useful. Since I first configured bartleby, I have set up a collection of relatively simple scripts to watch other services on our network and page me in the event of an irregularity.

A fall-back e-mail system is a good way to sneak Linux into a low profile but “mission-critical” application in your organization. Once you've proven Linux is a “real” operating system to any skeptical decision maker, you can begin to utilize it in higher-profile roles.

John Blair currently works as a software engineer at Cobalt Microserver. When he's not hacking Cobalt's cute blue Qube, he's hanging out with his wife Rachel and newborn son Ethan. John is also the author of Samba: Integrating UNIX and Windows, published by SSC. Feel free to contact him at jdblair@cobaltmicro.com.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #52, August 1998

Our experts answer your technical questions.

Ethernet Module Destruction

Why does running **sndconfig** get my Soundblaster 16 card recognized and up and running with the correct modules but destroys my Ethernet modules loaded by **kerneld**?

I have tried both my Intel Ether Express and SMC Ultra cards (not at the same time) and the same thing happens with both network cards as soon as sndconfig is run. Running the Kernel Daemon Configuration in X has no luck in getting the Ethernet module loaded again.

—Alan Kendall Red Hat

I'm not sure why it happens, but sndconfig basically just messes with /etc/conf.modules. You might want to back that file up, run sndconfig, backup the new file, and replace it with your old file. Then merge the sound config lines from the file sndconfig created into your original one. If that doesn't fix it, chances are you have hardware conflicts with your sound card and your Ethernet card. If you have further problems, e-mail sound-bugs@redhat.com.

—Donnie Barnes redhat@redhat.com

openwin Setup

How do I set up **openwin**? I have a Trident 9750 card and a Cybervision C70 monitor, and I have tried numerous times using **xf86config**, but I cannot get it to work. I am relatively new to Linux. Thanks for all the help.

—Jim Brewer Slackware 3.4

You should try the latest XFree86-3.3.2 server. According to some Usenet discussions, the 9750 was not supported before 3.3.2.

—Pierre Ficheux pierre@rd.lectra.fr

Distribution Questions

I'm getting ready to install a version of Linux. I'm leaning toward Slackware because I'm told it's very powerful and not very fancy. I'm going for more of the learning factor by installing it.

Anyway, I was wondering if someone could basically explain the correlation between the different distribution names and the kernels. Are the kernels universal for all the distributions, or are they built separately? To what degree, if any, are they compatible? If they're not, I may take a look at the available software before I install any one version.

Thank you very much.

—Brian Crowe

Free software (in the sense of freedom) is developed by independent programmers or research groups, who donate it to the user community (sort of —authors do retain a copyright on the programs). A distribution is an independent effort to put together all of the needed software to build up a complete operating system with applications and all that's needed. Usually, distributors modify some of the original packages to suit their view of the overall system. The kernel is one of the first packages you update; they are all compatible.

At application level, there is a lot of standardization between the different distributions (thanks to the “Filesystem Hierarchy Standard”, an accepted document), the only real difference being in the organization of their graphical tools to administer the system.

—Alessandro Rubini rubini@systemy.it

The distributions are built around the Linux kernel, but they each have their own way of doing installation, packaging and configuration. After one installs a particular distribution, they can download the newest kernel and install it themselves if necessary.

—Mark Bishop mark@tct-net.org

FTP Problem

When doing an FTP install and encountering an FTP problem, the window with some detail of the problem is immediately overwritten by the FTP setup window. How do I determine the problem and attempt to fix it?

—Mike Newsome Red Hat 5.0

The FTP install tries to get each package twice. Once that fails, it errors out. Apparently the error window is getting overwritten with a new setup window.

In general, if you got an error like that, it means you lost your connection to the FTP server. There aren't many other “errors” that can occur in the FTP install. FTP installs are not recommended over crowded networks (i.e., the Internet), because it's getting impossible to reliably move 200+ MB of data without some sort of problem. Handling error conditions created by this situation during the install of a complete operating system is very difficult. The FTP install does work very well over local or semi-local networks where you are guaranteed at least some bandwidth.

—Donnie Barnes redhat@redhat.com

Configuration Question

My notebook is using the CT65555 VGA chip set (2MB) with a 13.3-inch Active display (1024x768), and I can only get 1024x768 8-bit colour. How can I configure Red Hat to use 1024x768 16-bit colour?

—Corne Red Hat 5.0

First, edit your /etc/X11/XF86Config file and add another “Display” subsection to the “Screen” section. The subsection you add should look like this:

```
Subsection "Display"
 Depth 16
 Modes "1024x768"
 ViewPort 0 0
 Virtual 1024 768
 EndSubsection
```

Then run X this way to tell it to use the 16-bit depth:

```
startx -- -bpp 16"
```

—Scott Maxwells max@pacbell.net

Adobe Fonts, TeX and Linux

I am trying to buy an Adobe type 1 font (Caslon 224) for use with TeX on a Linux system. I called Adobe to order the font and was told, “We only sell products for Windows and Macintosh. We do not support UNIX”. I tried ordering the Mac version of the font. When the package arrived it had three floppies. Two of them hold Adobe's ATM program, and Hfsutils could mount each of them on my Linux system. The third holds Caslon, and I could not mount it with Hfsutils or in any other way. In short, I see no way to get either **pfa** or **pfb** files for Caslon. I've spent several days poking around the Web, and I can't find any references to this problem—let alone a solution.

I am nearly finished with a Linux application for an academic journal. It allows the journal to code articles, book reviews and other material in the journal. My program produces a TeX version of the journal, HTML for the parts that go on the journal's web site, and index files compatible with 60 years of index data for the journal. I started the project with the understanding (from Walsh's *Making TeX Work*) that TeX can use any Adobe type 1 font. Now, it appears that is only true on a Windows or Mac platform.

Am I missing something (perhaps really obvious)? If not, this looks to me like a serious problem for Linux users who want to develop and run commercial applications.

—David Bausum

I agree with you, this is a real problem for Linux users. Commercial software packages tend to assume the user is not able to copy files around, so they encapsulate every data file into a proprietary format, and distribute their own executable programs to extract such data.

The right approach would be to distribute zip files, or another known data format, but everyone thinks they have the best format around, and they don't care about real-world computer users.

Although I have never used Adobe fonts, I'm pretty sure that after installing them on one of the supported operating systems you'll find the needed .pfa and .pfb files, which will work for Linux. This means you must first access a Windows or a Macintosh box to do the initial install of the fonts, then copy them to your Linux box.

—Alessandro Rubini rubini@systemy.it

[As a side note, SSC has used the Windows versions of the Adobe fonts on Linux quite successfully on our Reference Cards. —Editor]

Easy Migration

I'm preparing to add a hard drive to my system. Is there a relatively easy way to migrate part of the file system to the new drive (/usr/bin) without a full backup or reinstall?

—Alan Jump Red Hat 5.0

You need to partition the new drive using Linux **fdisk**, changing the system ID flag for the partition(s). After that is done, you need to "make" new file systems, using **mkfs**. Mount the new file system on an empty directory such as /mnt. Then use **cpio -pdum** to copy the content of the old /usr/bin directory to the /mnt directory. Add a new line in the /etc/fstab file, indicating the new file system is to be mounted on the /usr/bin directory. Bring the machine to the single-user state, move the old /usr/bin to /usr/bin.old, and create a new, empty /usr/bin. Reboot, and the new file system will contain your /usr/bin files.

—Paulo J V Wollny paulo@wollny.com.br

Downloading Photos

I need a program, script or information on how to get the pictures off of my Kodak DC20 digital camera. Yes, I have a Windows 95 CD and the twain software that came with the camera and plenty of hard drive space for a partition, but do I have to put Windows 95 back on my nice Linux box? I am new to Linux, but I want to use only Linux if I can.

—Dick Colclasure Red Hat 5.0

Check out "Kodak DC20 Secrets" web page: http://home.t-online.de/home/oliver.hartmann/dc20secr.htm.

—Pierre Ficheux pierre@rd.lectra.fr

Unresolved Symbols

"Unresolved symbols in module" warnings at boot time are annoying me.

I've heard of several workarounds for this problem, but never found a final solution. I've recompiled my kernel several times, going through the Makefile to check that all kernel options are fine. I've followed the make sequence correctly (which includes the compilation and installation of all modules from scratch). Yet, I'm always surprised by at least one of these warnings at boot-time. Rather than a magical solution, I would appreciate a good description of the problem (which I'm sure will help me solve this issue).

Thank you guys. Keep up the good work.

—German Horacio

Kernel modules are stored in the /lib/modules directory in a subdirectory corresponding to the kernel release. So if you have just compiled kernel 2.0.33, the result of

```
make modules_install
```

would be to place all the kernel modules in /lib/modules/2.0.33. Unfortunately, if you already have a 2.0.33 subdirectory from a previous kernel compile or kernel install, these new modules will add to those already there. If some of the original modules were not compiled as modules or are not supported in the new kernel, they will generate an “unresolved symbols” error message when the new kernel is booted.

A safe solution is to always move the old modules directory to a neutral location, such as:

```
mv /lib/modules/2.0.33 /lib/modules/2.0.33.old
```

before executing **make modules_install**. It can be deleted later when you are sure you will not need the old kernel. This same error (and others) could be produced by failing to run

```
make modules
make modules_install
```

after a new kernel compile.

—Dwight Johnson dwj@aaronsrod.com

Answers published in Best of Technical Support are provided by a team of Linux experts.

Archive Index Issue Table of Contents

Advanced search