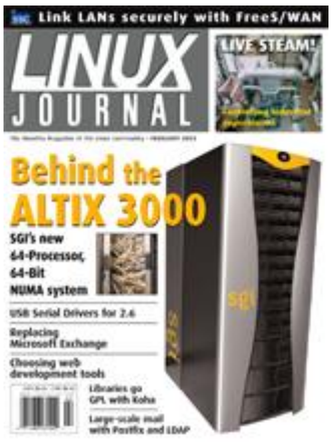
LINUX JOURNAL

Advanced search

Linux Journal Issue #106/February 2003



Koha: a Gift to Libraries from New Zealand

Pat Eyler

Issue #106, February 2003

Free software helps librarians serve the public on a reasonable budget.

The Maori word for a gift or donation is *koha*. It's also an integrated library system from New Zealand. Written for the Horowhenua Library Trust (HLT), it was licensed under the GPL and is now in use by libraries around the world.

History of the Project

In 1999, HLT made a momentous decision. They were using a 12-year-old integrated library system (ILS) that was no longer being developed. They knew the system wasn't Y2K-compliant, and they realized it no longer fit their needs. HLT also knew that buying a new system would cost them a lot of money up front and would require capital improvements they couldn't afford to make (communication lines and gear to support the new system).

Considering all of these factors, HLT, in consultation with Katipo Communications, decided to write their own system. They then decided to release this new system under the GPL, ensuring that other libraries could benefit from the work and also cooperate in future development of the system. This decision has had far-reaching effects.

Koha was developed during the fourth quarter of 1999 and went into production on January 1, 2000. There was a brief flurry of work on the system, and it was released to the world early that year. Koha won two awards in 2000: the 3M award for Innovation in Libraries and the ANZ Interactive Award (Community/Not-for-Profit Category).

Initially, Koha was picked up by other libraries in New Zealand (many of them hiring Katipo for support). One early adopter, Mike Mylonas, caught the vision of open-source software in libraries and began to contribute to the project.

Mike currently supports Koha for four private libraries, one for his current employer and three for nonprofit organizations.

It didn't take long for Koha to cross the Pacific. In the fall of 2000 the rural Coast Mountain school district in British Columbia, Canada, was looking for a solution for their library needs. They had been running a home-brew system built on Apple II computers, and it had finally died. Finding the money for a proprietary solution would be difficult (a small elementary school in New England recently received a quote for $20,000 to install a new ILS—proprietary library automation isn't cheap), so they put one of their network technicians to the task of finding a better option.

Steve Tonnesen, Coast Mountain's network engineer, came across Koha and started to evaluate it. It took him about two days to get Koha up and running. Once he had that base to work from, he starting hacking. He cleaned up the circulation interface, added importing tools and wrote a Z39.50 client for querying other libraries. Z39.50 is a standard protocol libraries use to exchange data about books. Word of this new option spread quickly, and he soon had three schools running the new system. Steve's changes went back into the main Koha system, and he became a member of the development team.

During April and May of 2002, Koha development took another big step. Project leadership always had come from Katipo, but the development team was now much more international and new development goals were being proposed. One of the first steps was the beginning of the 1.2 release cycle. These releases have focused on building basic functionality and greater stability. So far, there have been four releases in this series. New features include an installation script, a fully template-driven on-line public access catalog (OPAC), which supports both translation and customization and bundled user documentation.

Right now, development is running in earnest on the 1.4 series, which features a new database schema that supports several flavors of MAchine Readable Cataloging (MARC), the cataloging standard used by libraries. The first development release in this series (1.3.0) was made on September 24, 2002. A second release occurred in October, and a 1.4.0 release is expected to occur in the first quarter of 2003.

Using and Maintaining Koha

Koha is pretty undemanding as library systems go and runs handily on a stock Linux server. HLT is a library with 25,000 patrons at four locations and a collection of 80,000 items. They run over 1,200 transactions a day on a system with dual P3 1GHz processors and 1GB of RAM.

At the Immaculate Heart of Mary School library in Madison, Wisconsin, Robert Maynord installed Koha on an AMD 1800-based system with 256MB of RAM. Coast Mountain's systems run on 200MHz Pentiums with 64MB of RAM located in each school.

Getting Koha running in a library used to be a rather daunting task, but two easy methods now are available. The easiest method is to download the CD image, burn a copy with a CD burner and boot the new Koha server from the CD. You also can use the install script to set up Koha on your hardware.

The CD can be run as a demo system, using the included data, or it can be used as your server. If you choose to use it as your server, you will need to create a set of data files on your server's hard drive. The CD provides an interactive tool to do this.

If you'd rather install your own copy, the process is a bit more involved, but it still is not difficult. Before you get started, you should make sure some basic components are installed, namely Perl, Apache and MySQL. You'll need a few Perl modules as well, but the install script helps you take care of those. The install script has made installing Koha pretty painless. An upgrade script also has been written to help ease the burden of keeping the system up to date.

The Business Case for Libraries

Proprietary ILS packages are expensive beasts. A larger library may pay well in excess of $500,000 US for the server, clients and software, and it still has yearly license and support fees to worry about. Once a library has bought their system, they experience a high barrier to change as well. Data is most often kept in proprietary formats from which it is difficult to export—in some cases, the data is actually “owned” by the system vendor.

As with all non-free software, customers are left at the mercy of their vendors for enhancements and customizations. Library system vendors historically have been slow to provide innovative new options. Although user groups exist for many of the existing systems, they seem to be more like mutual support groups than sources of feedback for the vendors. A worse fate is in store for those whose ILS vendor goes out of business or is bought by another vendor.

This situation presents a great opportunity for free software. It's an opportunity that's not lost on librarians either, at least not all of them. There is still a great deal of ignorance and inertia to overcome. During an interview for the 2002 American Library Association's (ALA) presidential election this past spring, the candidates were asked what were their stands on free software. One of the responses boiled down to, “We need to support standards and let the vendors work out how to provide the solutions we need.”

More encouraging signs are emerging. The ALA has an information technology interest group, which reviewed open-source software in the March 2002 edition of their journal. Some of the articles were favorable, while others expressed a lack of confidence in free software's ability to produce a viable ILS. The definition of a viable ILS seems to vary considerably from library to library.

Some of the brightest lights come from the small, but growing open-source subculture, within the library community. The best example of this is the Open Source for Libraries group, hosted at www.oss4lib.org, which maintains a news site and mailing list. usr/lib/info (www.usrlib.info) is another group with a similar focus but a less technical approach.

A number of presentations made at library conferences in 2002 offered more encouraging signs. In October 2002, Chris Cormack (one of the original Katipo developers of Koha) was at the Ohio Library Conference to talk about what we've done and where we're going. Open-source software also got on the program at the Michigan Library Association Conference, the British Columbia Library Association Conference and Access 2002, a Canadian conference on internet-based technology for libraries.

Perhaps the greatest indicator of our success is that libraries have started to sponsor the development of features they feel are important. Sometimes this patronage is direct, as with Nelsonville Public Library, which contracted with one of the core Koha developers to finish work on its internal use of MARC. In other cases, the link is less direct, as is the case with recent work done by another developer being paid by a library automation vendor to develop a Koha solution for the vendor's customer.

Free Software and Librarians, a Natural Match

Librarians espouse many of the same ideals that drive the free software community. They collaborate and communicate; they work hard to share the results of their work with one another. They understand freedom and feel that it's an important value. That more librarians aren't actively using and evangelizing free software is an indictment against us for not letting them in on our secret.

It's important that we not think we'd be munificent benefactors, bringing a sack full of goodies to share. We can learn a lot from librarians as well. They have a number of skills that we, as a community, lack.

One of the key skills librarians bring to the table is information architecture. Librarians have spent a long time organizing information and making it accessible. If these skills could be harnessed in the free software community, we might see less duplication of efforts due to ignorance of existing projects, better cooperation between projects and easier acquisition of information by new developers and users.

Librarians also have been around as a group much longer, and they are fixtures in the academic community as well. This presence, and the established connections that come with it, could pay off handsomely if we used them to help spread the work of free software.

Librarians also could play a key role in creating and improving documentation in free software projects. Librarians tend to be good editors; they also have a good sense of what questions people tend to ask—the kind of thing you really want in your documentation. It also doesn't hurt that they have an expectation that documentation and support will be there. I have received personal mail (and phone calls) from librarians since my earliest involvement in Koha—writing good documentation becomes more attractive when you're faced with the alternative.

Librarians also are more likely to be involved in direct communication with end users than are most free software hackers. They are at the forefront of user-interface questions and internationalization issues. Putting that experience to work in creating user-friendly interfaces and documentation would be a great boon to most projects. Librarians also are much less forgiving of the technology; it has to run, all the time and within much tighter parameters than do a lot of other types of applications.

In recent years, open-source developers have become more political. Librarians still have a significant leg up on us in this area, however. What's more, their political ends overlap significantly with our own. Working together to help ensure open access to information, widespread adoption of free software and improved educational opportunities would be a win for both sides.

Finally, librarians tend to do a good job of engaging the public. They advertise to, interact with and provide services for our communities. They are seen as trusted sources of information, true public servants. If libraries become staunch bastions of the concepts behind free software, we gain a tremendous ally in reaching out to those who don't yet use or understand free software.

Looking Ahead

A lot of opportunities are on the horizon for Koha. As the 1.4 release series stabilizes, the developers' eyes have already targeted a number of new projects. One of the areas starting to get more attention is translations. French and German teams are already in place, and interest in Italian, South African and Spanish groups has started to build. (A nice side effect of this work is, we're gaining experience here that can help other free software projects.)

Right now, Koha is installed in libraries with collections of up to about 300,000 items. This is still on the small end of medium-sized libraries. Work to enable Koha to scale well past that is already on the drawing boards. The search tools are being rewritten to improve their efficiency and allow new searching options.

Work is also underway to build a reporting API and a number of bundled reports for Koha. These reports range from inventory checking, to financial reports and into more esoteric realms like “weeding”, or removing infrequently used items.

A bundled Z39.50 server and support for NCIP, a library protocol for handling interlibrary requests, are both in development and are being sponsored by libraries interested in using Koha. These new features will allow a Koha-based library to participate in larger library communities, including interlibrary loan programs, and to function as a solution for “union” catalogs, catalogs that integrate all of the libraries in a region.

Members of the Koha community have started a strategic discussion of what Koha needs to provide to continue to thrive. This project is hosted at www.kohalabs.com/projects/koha2010 and welcomes new participants.

So What Is an Integrated Library System (ILS)?

Resources

email: pate@eylerfamily.org

Pat Eyler is a Ruby, Perl and Linux geek. Currently he is the Kaitiaki (manager) of the Koha Project. When he's not playing with computers, he likes to read, cook and spend time with his kids.

Archive Index Issue Table of Contents

Advanced search

Understanding and Replacing Microsoft Exchange

Tom Adelstein

Issue #106, February 2003

The hardest-to-replace Microsoft server software is the expensive, frustrating Exchange. Here's how IBM and Bynari sent it packing.

Linux by itself provides a formidable set of internet applications for mainframes, which have always needed them. IBM's eServer strategy seemed incomplete without a robust set of internet tools, which it promised to provide to all of its brands. Near the end of calendar year 2000, IBM demonstrated it could host a thousand instances of Linux on a single S/390 mainframe.

Figure 1. One mainframe supports thousands of Linux instances.

Even so, IBM realized that web servers and GNU applications didn't provide a complete value proposition. IBM needed an application that made Linux a host that reached further into mainstream computing. So they made a call on us.

In April 2001, when the sales manager for IBM's zSeries, the new name for the S/390, visited our company, Bynari, Inc., I did not understand his interest. After his visit, I understood it perfectly. Bynari, Inc. became IBM's first Linux Influencer Partner.

E-Mail, the "Killer Application"

Initially, people knew us for making Linux and UNIX clients talk to Microsoft Exchange servers. Looking to broaden our market, I found the "Exchange Replacement HOWTO" by Johnson and Mead (www.bynari.net/whitepapers_howto.html). Using their work as a guide, we built a server for our Linux client and Microsoft Outlook. Our server and its Outlook Configuration Guide caught on with the reseller channel.

We didn't know anyone at IBM when we ported our server code to a Linux instance running on an S/390 Multiprise 3000. Jimmy Lee, then with Equant, provided the resources to see if we could do it. Gary Ernst of Equant configured the S/390 instance of Linux and provided assistance in getting our server to work.

As long as Microsoft Outlook had an Internet Mail Only mode and provided peer-to-peer folder sharing, we had a product that allowed UNIX and Outlook clients to schedule meetings and delegate calendar tasks. Our server scaled nicely, and we mimicked the Exchange global address list (GAL) while providing views of users' free/busy time and a decent administrative interface.

But then Microsoft released Office XP and made major changes in Outlook. Suddenly, our products needed server-side calendaring. We feared the growing appetite of IBM enterprise customers for a low-cost server solution for Outlook might wane. Therefore, we needed peer-to-peer calendar sharing in Outlook's Internet Mail Only mode, or we needed something on the server side.

Forty-Five Days to Create a Solution

With the January 2001 LinuxWorld Conference & Expo approaching, IBM continued marketing our server, somewhat blind to the needs of Outlook XP users. I knew we had to do something and do it fast or lose IBM's trust. At that time, the person who best understood the market problem was Roger Luca of Mainline Information Systems.

Fortunately, Roger and I developed a good working relationship. With Roger at the head of sales and marketing, Mainline became the largest reseller of IBM mainframes. They also were our biggest supporter outside IBM. Roger provided us with hardware resources to help us build server-side calendaring into our product, as well as with people to support us if we ran into hardware-related problems.

Surprising the Development Team

Imagine having completed an exhaustive year of development. You have trips scheduled for the holidays and other plans. Then you get a call on your cell phone; your boss asks you to attend an important meeting. That's what happened in our shop. I could hear the dread in the developers' voices when they answered my call.

We met on November 7, 2001, to see if we could deliver a server-side calendar solution by Christmas. Mainline had several sales pending, and they needed that functionality. Two of my people agreed to work with me to get the solution.

Technology Challenges Not for the Faint of Heart

As the senior developer, I provided the project framework. In theory, a project has a variety of phases and processes. To shorten the project's life, I instituted a three-step approach that called for research, invention and execution. I gave each phase a milestone. So instead of starting with code, we started browsing the Internet and any books we could find.

Following a week of intensive research, we discovered our challenges. We had to figure out how Microsoft's DCE-RFC protocol stored and moved calendar events around. We had to interpret the stored data, provide a format that we could store on an IMAP server and then forward the data to an Outlook client in its familiar schema. We also had to provide access control over those information stores to allow a user to appoint and delegate control over his or her calendar to other users with varying permission levels.

We spent another intense week researching and discovered a consensus. Every expert, newsgroup, Outlook specialist and company that tried said one could not create a Microsoft Outlook calendar store on an IMAP server. Here's how we did it.

Forget the Transport and Focus on the Data

First, we observed that Outlook did around 95% of the work in the Microsoft Exchange model. Exchange, as its name implies, transfers data between or among Outlook users. Even so, Exchange controlled the transport protocol and that presented a problem. People use the term MAPI to describe the Exchange protocol for transferring data among Outlook users. I don't see MAPI as being the appropriate term.

When intercepting Outlook messages, we discovered large arrays of binary data instead of text. I recognized the data but didn't realize where I had seen it before.

The Different Faces of Outlook

Outlook runs in two different modes: Corporate Workgroup mode and Internet Mail Only mode. In the Corporate Workgroup mode, Microsoft turns on all of Outlook's highly regarded features. In the Internet Mail Only mode, Microsoft uses a completely different and undocumented application program interface (API) with a limited feature set. Without an Exchange server, Outlook doesn't function in the Corporate Workgroup mode at all and has a limited set of features.

In Outlook's Workgroup mode, or when connected to an Exchange server, its data moves around in binary form. That binary data becomes impossible to recognize by an uninformed observer.

While researching, I found a developers' site on SourceForge.net that was porting Open DCE to Linux. I e-mailed one of the developers who wrote back and mentioned that the Open Group contributed the code.

I went to the Open Group's web site, searched the archives and found an old article mentioning that Microsoft had licensed DCE. We downloaded the Open DCE code and, using the engine, shook hands with Outlook and then Exchange. We knew more about the transport protocol as a result. We also understood the presence of binary data streams.

What we discovered is Microsoft uses the distributed computing environment (DCE) as its transport when using Exchange and Outlook in the Corporate Workgroup mode. Microsoft provides a programming interface on top of DCE, which it calls MAPI. Still, underneath MAPI exists an open standards-based protocol (DCE), which Microsoft bought from the Open Group and modified.

One of the default functions in DCE automatically translates ASCII text into binary objects. Microsoft leaves the binary object undocumented. So most of the MAPI properties programmers tag wind up as binary code they would not recognize. To make matters a little more complex, Microsoft embeds the binary property code in a large array of null binary data, thus hiding it.

We began to understand the transport, but we realized that Outlook sent MIME attachments to other Outlook clients. Those attachments did not transform themselves into binary data. We concluded that Outlook also used encapsulation to pass attachments around, which led us to the TNEF object.

TNEF

Microsoft Exchange uses a number of programs it calls “service providers” that Linux users might call dæmons. Exchange service providers handle objects, which have state and behavior.

Transport neutral encapsulation format (TNEF) is a method to pass ASCII text, other files and objects, along with binary message data. The binary message data makes up the bulk of each TNEF object. TNEF encapsulates MAPI properties into a binary stream that accompanies a message through transports and gateways. Outlook can decode the encapsulation to retrieve all the properties of the original message. The TNEF object hides in MIME as an attachment.

When we found the properties, which created calendar events, we built a TNEF encoder and soon began sending calendar events to and from Outlook clients with SMTP. We immediately recognized that we could use internet transport protocols and turn on Microsoft's Corporate Workgroup mode without MAPI. We knew we had arrived when we saw Microsoft Knowledge Base Article Q197204, which says that Microsoft does not support our transport protocol in the Workgroup mode.

Exchange Client Extensions

With our primary goal being server-side calendaring, we needed to create a message store to hold our Outlook client objects. As we used an IMAP server, we needed IMAP support, which Microsoft did not provide in the Workgroup mode. So we had to find a way to add IMAP client support to Outlook. The facilitator for adding functionality to an existing Outlook client involved what most people think of as a plugin.

When Microsoft first released Exchange, Outlook didn't exist. Instead, Microsoft provided a set of Exchange messaging clients for its different Windows operating systems. Microsoft also provided an extensible architecture for those Exchange messaging clients. Client extensions provided developers with a way to change the default behavior of the Exchange client. When Microsoft released Outlook, it continued providing support for the Exchange client extension architecture for compatibility with existing client extension DLLs.

Client extensions allow one to alter the default behavior of the client. Microsoft saw the advantages of an extension as a convenient way to integrate customized features and new behavior directly into the client, instead of having to write a separate MAPI application. We, however, saw extensions as a way to add IMAP client services to Outlook in the Workgroup mode. Using this architecture, we added commands to Outlook menus, custom buttons to the toolbars and the ability to preprocess outgoing and incoming messages with IMAP client services.

Luckily, we had already written client libraries for IMAP when we built our Linux client the previous year. We simply needed to port them to Windows. Our familiarity with the function calls, headers and protocols reduced our overall effort.

Once we built a Microsoft DLL for the client functions, we added it as an Outlook extension. Luckily, it worked the first time we tried it. By choosing the rich text format (RTF) for mail and meeting invitations, our TNEF objects attached themselves to the messages. Because Outlook created the TNEF objects, it exchanged them without any problems.

At this point, we uploaded our messages to our IMAP folders using the Microsoft .pst file as store and swap space. Staying connected to Exchange and using our server for message stores, we noticed compatibility between the two systems. We dragged and dropped objects from the Exchange folders into our IMAP folders. By doing this we discovered that tasks, journal entries, calendar events and so on, all showed up in Outlook as if they arrived from Exchange. The calendar also worked perfectly.

Exchange

When you look at Exchange and study its components, you find they number only four. The first is an information store or message store. The store holds individual user messages and has an access control list (ACL) engine associated with them. Similar to RFC-compliant IMAP servers, namespace differs according to whether the stores belong to individual users or whether the folders are public. Microsoft uses an Access database for storing message stores. The limitation of Microsoft's Jet Engine technology and the Access MDB file prevents vertical scalability.

Secondly, Exchange has a directory. Microsoft structured their Exchange directory with object classes and attributes. The Exchange directory structure resembles the RFC-compliant LDAP protocol. However, Microsoft added Object Classes and changed the attribute names within those and other classes.

Next, Exchange has a mail transfer agent or MTA. Microsoft's MTA appears similar to the MTA used in an earlier product called Microsoft Mail 3.5. The Microsoft Mail MTA requires connectors or gateways, which rewrite their proprietary mail headers to those that comply with foreign systems, such as Lotus Notes, X-400 and RFC 822 internet mail standards. Unlike sendmail and similar internet MTAs, Exchange's MTA lacks configuration options.

Finally, Exchange has a component called a system attendant. The attendant handles every action taken within Exchange, from sending and receiving e-mail to filling requests for addresses from the Exchange directory. In many ways the system attendant resembles an attempt to provide interprocess communication (IPC), which Microsoft's operating systems lack.

Out-Scaling Microsoft Using Berkeley DB

Our Linux server-side solution included similar components to those found in Exchange. The first is the Cyrus IMAP message store. Cyrus stores hold individual user messages and have an ACL engine associated with them. Namespace differs according to whether the stores belong to individual users or whether the folders are public. Cyrus uses the Berkeley Database from Sleepycat Software. Where Microsoft's Jet Engine and Access database technology prevents scaling, Berkeley DB's high performance and scalability support thousands of simultaneous users working on databases as large as 256 terabytes.

Secondly, Linux has a directory. While Microsoft structured their Exchange directory to resemble the Lightweight Directory Access Protocol (LDAP), the Linux solution uses OpenLDAP software, an open-source implementation of LDAP. To accommodate Outlook clients, we added the Exchange object classes and their noncompliant attribute names. We indexed the Microsoft-based distinguished names and created a high-performance global address list.

Like Exchange, the Linux solution has an MTA that can be managed and configured internally and doesn't need external connectors. The University of Cambridge developed the Linux MTA we use, called Exim. Exim has numerous configuration options, including file lookups, local delivery and regular expression support. In the context of the Linux MTA, users provide regular expressions to filter content coming in and going out.

Replacing Exchange

In the “Exchange Replacement HOWTO”, Johnson and Mead leave the tasks of adding server-side messaging and the administrative console to the next generation of Linux developers. In this article, we explain how one could transform Exchange transports and message stores. We accomplish this in two steps. First, we capture Outlook messages and decode their TNEF objects. Secondly, we use the Exchange client extension architecture to add IMAP functionality to Outlook in its Corporate Workgroup mode.

These two steps can allow a programmer or a seasoned administrator to create an alternative service provider for Outlook and serve a number of conventional mail clients. Linux mail servers do not discriminate based on the platform one uses. One can use Netscape Mail, Outlook Express, Ximian Evolution, mutt or Pine, to mention a few of the available MUA.

Highly scalable Linux components, such as Cyrus IMAP, OpenLDAP and Exim, can replace dozens of Exchange servers on a single Intel platform. The layers of interfaces and outdated DCE components used by Exchange do not hinder Linux. With Linux on the zSeries mainframe, we can replace hundreds of Exchange servers.

If you're looking for a graphical administrative console, projects such as PHP Cyrus tools, cyrus_imap-sql, Webmin and Replex can make administration of the server a simple task.

In general, few people would consider replacing Exchange with Linux an easy task. In spite of that, our development team proved that it could be done. Hopefully, we have taken much of the mystery and intimidation out of the Exchange server.

Resources

email: adelste@netscape.net

Tom Adelstein works for Xandros, Inc. and heads up the company's server division in Dallas, Texas. His current interest lies in the field of web services and supporting Xandros Linux Desktop. Tom welcomes your comments at tadelstein@xandros.com.

Archive Index Issue Table of Contents

Advanced search

Scaling Linux to New Heights: the SGI Altix 3000 System

Steve Neuner

Issue #106, February 2003

With 64 processors and 512GB of memory, SGI claims the title of world's most powerful Linux system.

SGI recently debuted its new 64-bit, 64-processor, Linux system based on the Intel Itanium 2 processor—a significant announcement for the company and for Linux. This system marks the opening of a new frontier as scientists working on complex and demanding high-performance computing (HPC) problems can now use and deploy Linux in ways never before possible. HPC environments continually push the limits of the operating system by requiring larger numbers of CPUs, higher I/O bandwidth and faster and more efficient parallel programming support.

Early on in the system's development, SGI made the decision to use Linux exclusively as the operating system for this new platform. It proved to be a solid and very capable operating system for the technical compute environments that SGI targets. With the combination of SGI NUMAflex global shared-memory architecture, Intel Itanium 2 processors and Linux, we were breaking records long before the system was introduced.

The new system, called the SGI Altix 3000, has up to 64 processors and 512GB of memory. A future version will offer up to 512 processors and 4TB. In this article, we explore the hardware design behind the new SGI system, describe the software development involved to bring this new system to market and show how Linux can readily scale and be deployed in the most demanding HPC environments.

Hardware and System Architecture Background

The SGI Altix 3000 system uses Intel Itanium 2 processors and is based on the SGI NUMAflex global shared-memory architecture, which is the company's implementation of a non-uniform memory access (NUMA) architecture.

NUMAflex was introduced in 1996 and has since been used in the company's renowned SGI Origin family of servers and supercomputers based on the MIPS processor and the IRIX 64-bit operating system. The NUMAflex design enables the CPU, memory, I/O, interconnect, graphics and storage to be packaged into modular components, or bricks. These bricks can then be combined and configured with tremendous flexibility to match a customer's resource and workload requirements better. Leveraging this third-generation design, SGI was able to build the SGI Altix 3000 system using the same bricks for I/O (IX- and PX-bricks), storage (D-bricks) and interconnect (router bricks/R-bricks). The primary difference in this new system is the CPU brick (C-brick), which contains the Itanium 2 processors. Figure 1 shows the types of bricks used on the SGI Altix 3000 system. Figure 2 depicts how these bricks can be combined into two racks to make a single-system-image 64-processor system.

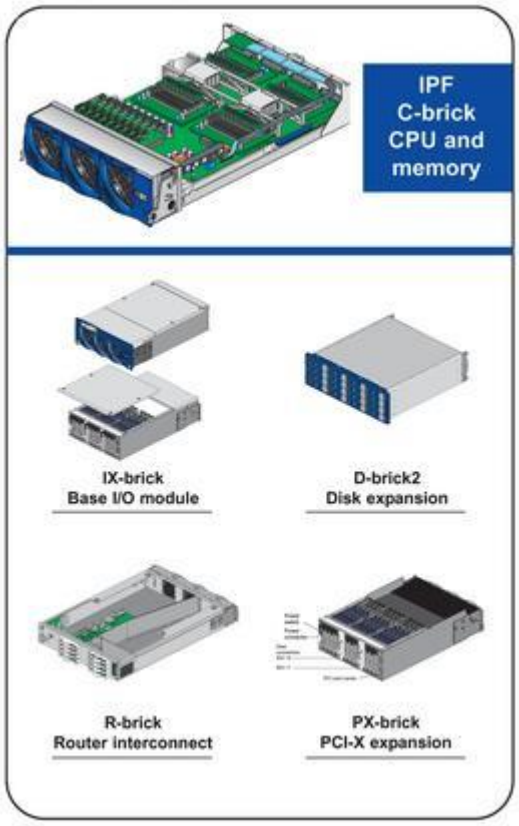


Figure 1. NUMAflex Brick Types

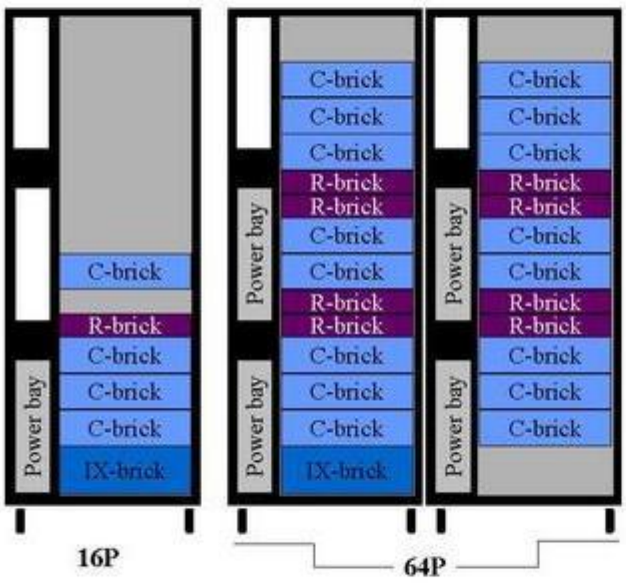


Figure 2. Two Possible NUMAflex Configurations

SGI Altix 3000 C-Brick Block Diagram and Specifications

Preparing Linux for a New Hardware Platform

On a well-designed and balanced hardware architecture such as NUMAflex, it is the operating system's job to ensure that users and applications can fully exploit the hardware without being hindered due to inefficient resource management or bottlenecks. Achieving balanced hardware resource management on a large NUMA system requires starting kernel development long before the first Itanium 2 processors and hardware prototype systems arrive. In this case, we also used the first-generation Itanium processors for making the CPU scaling, I/O performance and other changes to Linux necessary for demanding HPC environments.

The first step in preparing the software before the prototype hardware arrives is identifying, as best you can, the necessary low-level hardware register and machine programming changes the kernel will need for system initialization and runtime. System manufacturers developing custom ASICs for highly advanced systems typically use simulation software and tools to test their hardware design. Before hardware was available, we developed and used simulators extensively for both the system firmware and kernel development to get the system-level software ready.

When the original prototype hardware based on first-generation Itanium processors arrived, it was time for power-on. One of the key milestones was powering the system on for the first time and taking a processor out of reset, then fetching and executing the first instructions from PROM.

Figure 3. SGI engineers celebrate power-on success.

After power-on, the fun really began with long hours and weekends in the hardware “bring-up” lab. This is where hardware, diagnostic and platform-software engineers worked together closely to debug the system and get the processor through a series of important milestones: PROM to boot prompt, Linux kernel through initialization, reading and mounting root, reaching single-user mode and then going into multi-user mode and then connecting to the network. After that, we did the same thing all over again with multiple processors and multiple nodes—typically pursued in parallel—with several other bring-up teams at other stations that trail closely behind the lead team's progress.

Figure 4. During bring-up, a hardware engineer, a PROM engineer and an OS engineer discuss a bug.

Once we had Linux running on the prototype systems with first-generation Itanium processors, software engineers could proceed with ensuring that Linux ran and, in particular, scaled well on large NUMA systems. We built and used numerous in-house, first-generation Itanium-based systems to help ensure that

Linux performed well on large systems. By early 2001, we had succeeded in running a 32-processor Itanium-based system—the first of its kind.

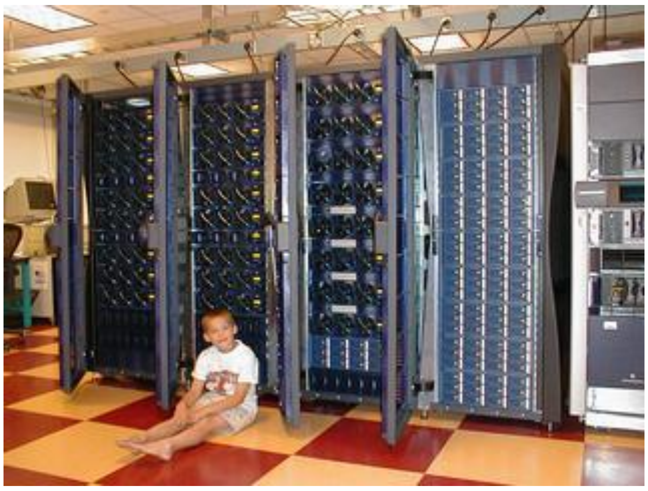


Figure 5. The author's son in front of an early 32-processor Itanium-based system, Summer 2001.

These first-generation Itanium-based systems were key in having Linux ready for demanding HPC requirements. Well before the first Itanium 2 processors were available from Intel, the bulk of the scaling, I/O performance and other changes for Linux could be developed and tested.

As one group of SGI software engineers was busy working on performance, scaling and other issues, using prototypes with first-generation Itanium processors, another team of hardware and platform-software engineers was getting the next-generation SGI C-brick with Itanium 2 processors ready for power-on to repeat the bring-up process all over again.

Figure 6. First power-on of the Itanium 2-based C-brick.

By mid-2002, the bring-up team had made excellent progress, from power-on of a single processor to running a 64-processor system. The 64-processor system with Itanium 2 processors again marked the first of its kind. All this, of course, was with Linux running in a single-system image.

Throughout this whole process, we passed any changes in Linux or bugs found back to the kernel developers for inclusion in a future release of Linux.

A Closer Look at Linux on Big Iron

Other Linux developers often ask, “What kind of changes did you have to make to get Linux to run on that size system?” or “Isn't Linux CPU scaling limited to eight or so processors?” Answering these questions involves examining further what SGI is using as its software base, the excellent changes made by the community and the other HPC-related enhancements and tools provided by SGI to help make Linux scale far beyond the perceived limit of eight processors.

On the SGI Altix 3000 system, the system software consists of a standard Linux distribution for Itanium processors and SGI ProPack, an overlay product that provides additional features for Linux. SGI ProPack includes a newer 2.4-based Linux kernel, HPC libraries highly tuned to exploit SGI's hardware, NUMA tools and drivers.

The 2.4-based Linux kernel used on the SGI Altix 3000 system consists of the standard 2.4.19 kernel for Itanium processors (kernel.org), plus other improvements. These improvements fall into one of three categories: general bug fixes and platform support, improvements from other work occurring within the Linux community and SGI changes.

The first category of kernel changes is simply ongoing fixes to bugs found during testing and the continued improvements for the underlying platform and NUMA support. For these changes, SGI works with the kernel team's designated maintainer to get these changes incorporated back into the mainline kernel.

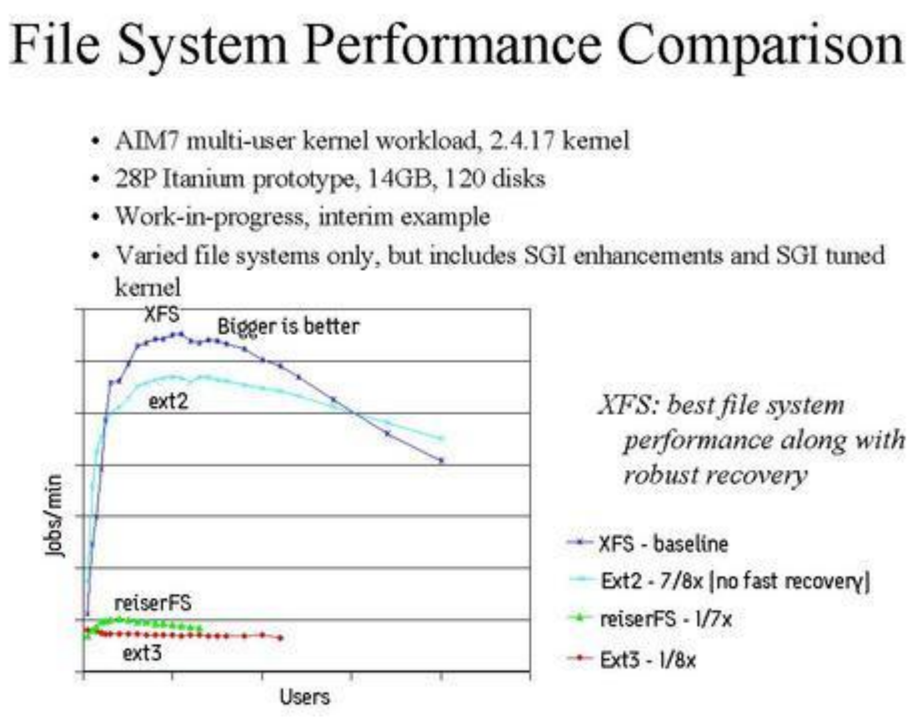
The second category of kernel improvements consists of the excellent work and performance patches developed by others within the community that have not been accepted officially yet or were deferred until the 2.5 development stream. These improvements can be found on the following VA Software SourceForge sites: “Linux on Large Systems Foundry” (large.foundries.sourceforge.net) and the “Linux Scalability Effort Project” (sourceforge.net/projects/lse). We used the following patches from these projects: CPU scheduler, Big Kernel Lock usage reduction improvements, dcache_lock-usage reduction improvements based on the Read-Copy-Update spinlock paradigm and xtime_lock (gettimeofday) usage reduction improvements based on the FRlock locking paradigm.

We also configured and used the Linux device filesystem (devfs, www.atnf.csiro.au/people/rgooch/linux/docs/devfs.html) on our systems to handle large numbers of disks and I/O busses. Devfs ensures that device path names persist across reboots after other disks or controllers are added or removed. The last thing a system administrator of a very large system wants is to have a controller go bad and have some 50 or more disks suddenly renumbered and renamed. We have found devfs to be reliable and stable in high-stress system environments with configurations consisting of up to 64 processors with dozens of fibre channel loops with hundreds of disks attached. Devfs is an optional part of the 2.4 Linux kernel, so a separate kernel patch was not needed.

The third category of kernel change consists of improvements by SGI that are still in the process of getting submitted into mainline Linux, were accepted after 2.4 or will probably remain separate due to the specialized use or nature of the patch. These open-source improvements can be found at the “Open Source at SGI” web site (oss.sgi.com). The improvements we made included: XFS filesystem software, Process AGGregates (PAGG), CpuMemSets (CMS), kernel debugger (kdb) and a Linux kernel crash dump (lkcd).

In addition, SGI included its SCSI subsystem and drivers ported from IRIX. Early tests of the Linux 2.4 SCSI I/O subsystem showed that our customers' demanding storage needs could not be met without a major overhaul in this area. While mainstream kernel developers are working on this for a future release, SGI needed an immediate fix for its 2.4-based kernel, so the SGI XSCSI infrastructure and drivers from IRIX were used as an interim solution.

Figures 7-9 illustrate some of the early performance improvements that were achieved with Linux on the SGI Altix 3000 system using the previously described changes. Figure 7 compares XFS to other Linux filesystems. (Note, for a more detailed study on Linux filesystem performance, see “Filesystem Performance and Scalability in Linux 2.4.17”, 2002 USENIX Annual Technical Conference, which is also available at oss.sgi.com). Figure 8 compares XSCSI to SCSI in Linux 2.4, and Figure 9 shows CPU scalability using AIM7.

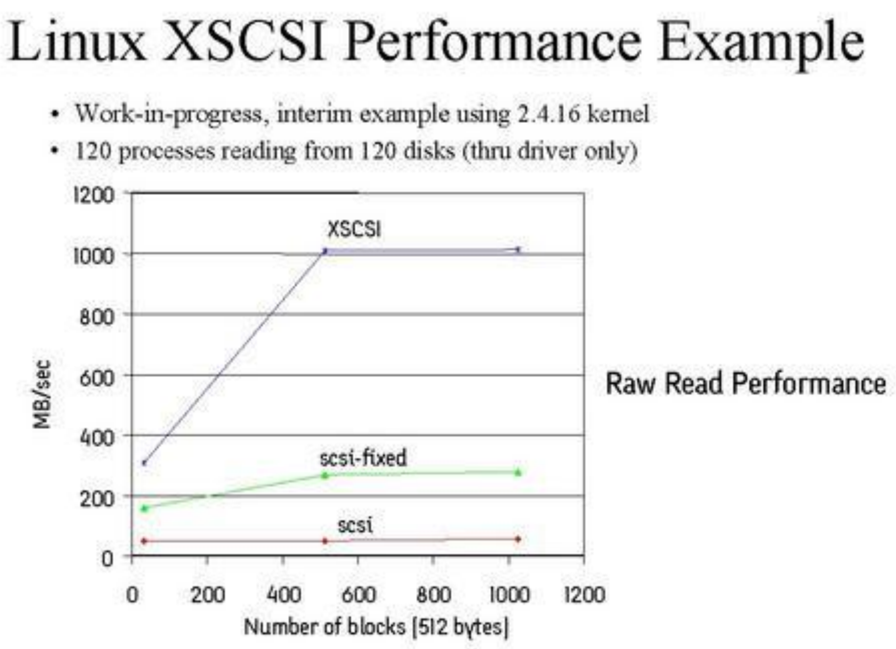


File System Performance Comparison

- AIM7 multi-user kernel workload, 2.4.17 kernel
- 28P Itanium prototype, 14GB, 120 disks
- Work-in-progress, interim example
- Varied file systems only, but includes SGI enhancements and SGI tuned kernel

XFS: best file system performance along with robust recovery

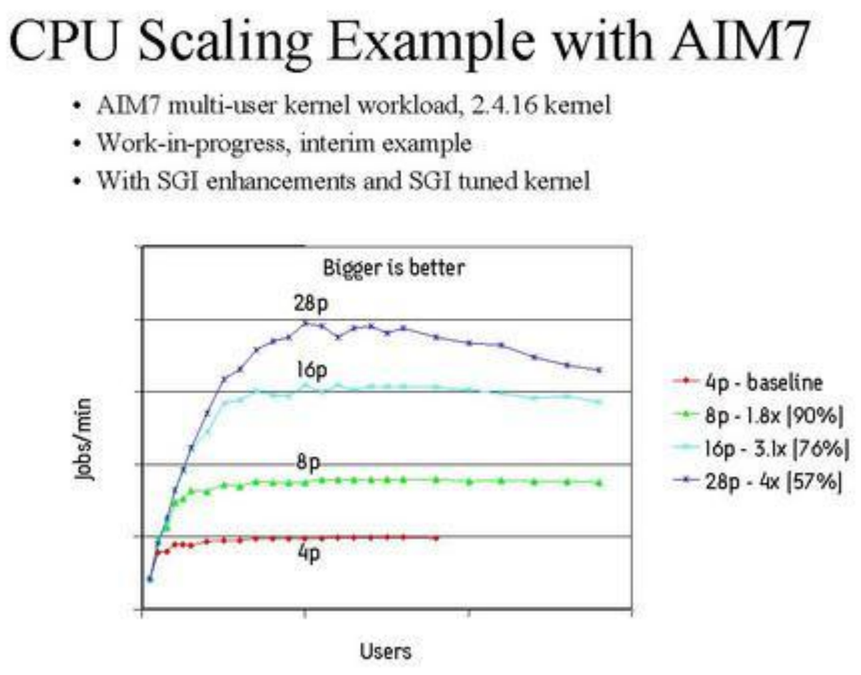
Figure 7. Filesystem performance comparison: AIM7 multi-user kernel workload, 2.4.17 kernel; 28 P Itanium prototype, 14GB, 120 disks; work-in-progress, interim example; varied filesystems only, but includes SGI enhancements and SGI tuned kernel.



Linux XSCSI Performance Example

- Work-in-progress, interim example using 2.4.16 kernel
- 120 processes reading from 120 disks (thru driver only)

Figure 8. Linux XSCSI performance example: work-in-progress, interim example using 2.4.16 kernel; 120 processes reading from 120 disks (through driver only).


CPU Scaling Example with AIM7

- AIM7 multi-user kernel workload, 2.4.16 kernel
- Work-in-progress, interim example
- With SGI enhancements and SGI tuned kernel

Figure 9. CPU scaling example with AIM7: AIM7 multi-user kernel workload, 2.4.16 kernel; work-in-progress, interim example; SGI enhancements and SGI-tuned kernel.

While SGI is focused more toward high-performance and technical computing environments—where the majority of CPU cycles is typically spent in user-level code and applications instead of in the kernel—the AIM7 benchmark does show that Linux can still scale well with other types of workloads common in enterprise environments. For HPC application performance and scaling examples for Linux, see the Sidebar "Already Solving Real-World Problems".

Figure 10 shows the scaling results achieved on an early SGI 64-processor prototype system with Itanium 2 processors running the STREAM Triad benchmark, which tests memory bandwidth. With this benchmark, SGI demonstrated near-linear scalability from two to 64 processors and achieved over 120GB per second. This result marks a significant milestone for the industry by setting a new world record among a microprocessor-based system, which was achieved running Linux within a single-system image! This impressive result also demonstrates that Linux can indeed scale well beyond the perceived limitation of eight processors. For more information on STREAM Triad, see www.cs.virginia.edu/stream.

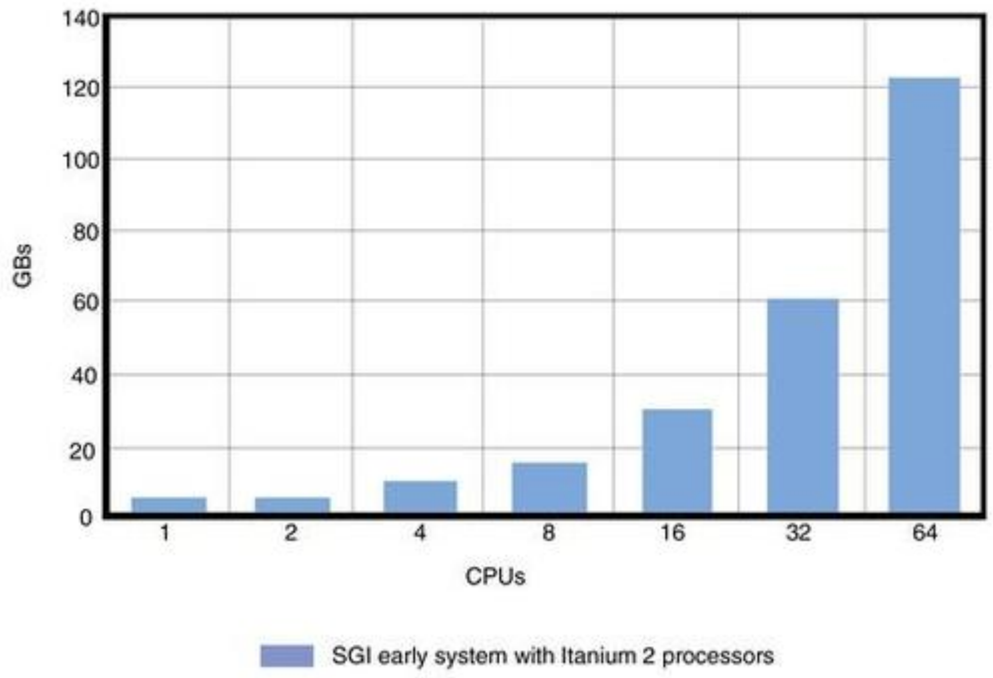


Figure 10. Near-linear STREAM Triad scalability up to 64 processors.

When you look at the list of kernel additions included in SGI ProPack the list is actually surprisingly small, which speaks highly of Linux's robust original design. What is even more impressive is that many of these and other changes are already in the 2.5 development kernel. At this pace, Linux is quickly evolving as a serious HPC operating system.

Already Solving Real-World Problems

Other Enhancements to Linux for HPC

SGI ProPack also includes several tools and libraries to help improve performance on large NUMA systems for solving a complex problem with an application that needs large numbers of CPUs and memory, or when multiple applications are running simultaneously within the same large system. On Linux, SGI provides the commands **cpuset** and **dplace**, which give predictable and improved CPU and memory placement control for HPC applications. These tools help unrelated jobs carve out and use the resources they each need without getting into each other's way or help prevent a smaller job from inadvertently thrashing across a larger pool of resources than it can effectively use. Therefore system resources are used efficiently and deliver results in a consistent time period—two characteristics critical to HPC environments.

Also, the SGI Message Passing Toolkit (MPT) in SGI ProPack provides industry-standard message passing libraries optimized for SGI computers. MPT contains MPI and SHMEM APIs, which transparently utilize and exploit the low-level capabilities within the SGI hardware, such as its block transfer engine (BTE) for fast memory-to-memory transfers and the hardware memory controller's fetch operation (fetchop) support. Fetchop support enables direct communication and synchronization between multiple MPI processes while eliminating the overhead associated with system calls to the operating system.

The SGI ProPack NUMA tools, HPC libraries and additional software support layered on top of a standard Linux distribution provide a powerful HPC software environment for big compute and data-intensive workloads. Much like a custom ASIC on hardware providing the “glue logic” to leverage and use commodity processors, memory and I/O parts, SGI ProPack software provides the “glue logic” to leverage the Linux operating system as a commodity building block for large HPC environments.

Conclusion

No one believed Linux could scale so well, so soon. By combining Linux with SGI NUMAflex system architecture and Itanium 2 processors, SGI has built the world's most powerful Linux system. Bringing the SGI Altix 3000 system to market involved a tremendous amount of work, and we consider it to be only the beginning. The aggressive standards-based strategy that SGI has for using Linux on Itanium 2-based systems is raising the bar on what Linux can do while providing customers an exciting, no-compromises alternative for large HPC servers and supercomputers. SGI engineers—and the entire company for that matter—are fully committed to building on Linux capabilities and pushing the envelope even further to bring more exciting breakthroughs and opportunities for the Linux and HPC communities.

Steve Neuner has been working in UNIX kernel development for the past 19 years at major computer manufacturers including MAI Basic Four, Sequent Computer Systems, Digital Equipment Corporation and SGI. Now with SGI, Steve is the Linux engineering director and has been working on Linux and Itanium-based systems since joining SGI four years ago.

Inside the Intel Compiler

Dale Schouten

Xinmin Tian

Aart Bik

Milind Girkar

Issue #106, February 2003

How did Intel's compiler beat gcc on Benchmarks? Intel's compiler developers explain IA-32 optimizations they use.

The increasing acceptance of Linux among developers and researchers has yet to be matched by a similar increase in the number of available development tools. The recently released Intel C++ and Fortran compilers for Linux aim to bridge this gap by providing application developers with highly optimizable compilers for the Intel IA-32 and Itanium processor families. These compilers provide strict ANSI support, as well as optional support for some popular extensions. This article focuses on the optimizations and features of the compiler for the Intel IA-32 processors. Throughout the rest of this article, we refer to the Intel C++ and Fortran compilers for Linux on IA-32 collectively as “the Intel compiler”.

The Intel compiler optimizes a program at all levels, from high-level loop and interprocedural optimizations to standard compiler data flow optimizations, in addition to efficient low-level optimizations, such as instruction scheduling, basic block layout and register allocation. In this article, we mainly focus on compiler optimizations unique to the Intel compiler. For completeness, however, we also include a brief overview of some of the more traditional optimizations supported by the Intel compiler.

Traditional Compiler Optimizations

Decreasing the number of instructions that are dynamically executed and replacing instructions with faster equivalents are perhaps the two most obvious ways to improve performance. Many traditional compiler optimizations fall into this category: copy and constant propagation, redundant expression elimination, dead code elimination, peephole optimizations, function inlining, tail recursion elimination and so forth.

The Intel compiler provides a rich variety of both types of optimizations. Many local optimizations are based on the static-single-assignment (SSA) form. Redundant (or partially redundant) expressions, for example, are eliminated according to Chow's algorithm (see Resource 6), where an expression is considered redundant if it is unnecessarily calculated more than once on an execution path. For instance, in the statement:

```
x[i] += a[i+j*n] + b[i+j*n];
```

the expression i+j*n is redundant and needs to be calculated only once. Partial redundancy occurs when an expression is redundant on some paths but not necessarily all paths. In the code:

```
if (c) {
    x = y+a*b;
} else {
    x = a;
}
z = a*b;
```

the expression a*b is partially redundant. If the else branch is taken, a*b is only calculated once; but if the then branch is taken, it is calculated twice. The code can be modified as follows:

```
t = a*b;
if (c) {
    x = y+t;
} else {
    x = a;
}
z = t;
```

so there is only one calculation of a*b, no matter which path is taken.

Clearly, this transformation must be used judiciously as the increase in temporary values, ideally stored in registers, can increase lifetimes and, hence, register pressure. An algorithm similar to Chow's algorithm (see Resource 9) is used to eliminate dead stores, in which a store is succeeded by another store to the same location before a fetch, and partially dead stores, which are dead along some but not necessarily all paths. Other optimizations based on the SSA form are constant propagation (see Resource 7) and the propagation of conditions. Consider the following example:

```
if (x>0) {
    if (y>0) {
        . . .
        if (x == 0) {
            . . .
        }
        }
    }
```

Since x>0 holds within the outmost if, unless x is changed, we know that x != 0, and therefore the code within the inner if is dead. Although this and the previous example may seem contrived, such situations are actually quite common in the presence of address calculations, macros or inlined functions.

Powerful memory disambiguation (see Resource 8) is used by the Intel compiler to determine whether memory references might overlap. This analysis is important to enhance, for instance, register allocation and to enable the detection and exploitation of implicit parallelism in the code, as discussed in the following sections. The Intel compiler also provides extensive interprocedural optimizations, including manual and automatic function inlining, partial inlining where only the hot parts of a routine are inlined, interprocedural constant optimizations and exception-handling optimizations. With the optional “whole program” analysis, the data layout of certain data structures, such as COMMON BLOCKS in Fortran, may be modified to enhance memory accesses on various processors. For example, the data layout could be padded to provide better data alignment. In addition, in order to make decisions that are more intelligent about when and where to inline, the Intel compiler relies on two types of profiling information: static profiling and dynamic profiling. Static profiling refers to information that can be deduced or estimated at compile time. Dynamic profiling is information gathered from actual executions of a program. These two types of profiling are discussed in the next section.

Profiling Optimizations

First, we will look at static profiling. Consider the following code fragment:

```
    g();
    for (i=0; i<10; i++) {
        g();
    }
```

Obviously, the call inside the loop executes ten times more often than the call outside the loop. In many cases, however, there is no way to make a good estimate. In the following code:

```
    for (i=0; i<10; i++) {
        if (condition) {
            g();
        } else {
            h();
        }
    }
```

it is difficult to say whether one condition is more likely to occur than another. If h() happened to be an exit or some other routine that was known not to return, it would be safe to assume the then branch was more likely taken and inlining g() may be worthwhile. Without such information, however, the decision of whether to inline one call or the other (or both) gets more complicated. Another option is to use dynamic profiling.

Dynamic profiling gathers information from actual executions of a program. This allows the compiler to take advantage of the way a program actually runs in order to optimize it. In a three-step process, the application is first built with profiling instrumentation embedded in it. Then the resulting application is run with a representative sample (or samples) of data, which yields a database for the compiler to use in a subsequent build of the application. Finally, the information in this database is used to guide optimizations such as code placement or grouping frequently executed basic blocks together, function or partial inlining and register allocation. Register allocation in the Intel compiler is based on graph fusion (see Resource 5), which breaks the code into regions. These regions are typically loop bodies or other cohesive units. With profile information, the regions can be selected more effectively and are based on the actual frequency of the blocks instead of syntactic guesses. This allows spills to be pushed into less frequently executed parts of the program.

Intra-Register Vectorization

Exploiting parallelism is an important way to increase application performance in modern architectures. The Intel compiler can be key in the effort to exploit potential parallelism in a program by facilitating such optimizations as automatic vectorization, automatic parallelization and support for OpenMP directives. Let's look at the automatic conversion of serial loops into a form that takes advantage of the instructions provided by the Intel MMX technology or SSE/SSE2 (Streaming-SIMD-extensions), a process we refer to as “intra-register vectorization” (see Resource 1). For example, given the function:

```
void vecadd(float a[], float b[], float c[], int n)
{
  int i;
  for (i = 0; i < n; i++) {
      c[i] = a[i] + b[i];
  }
}
```

the Intel compiler will transform the loop to allow four single-precision floating-point additions to occur simultaneously using the addps instruction. Simply put, using a pseudo-vector notation, the result would look something like this:

```
for (i = 0; i < n; i+=4) {
    c[i:i+3] = a[i:i+3] + b[i:i+3];
}
```

A scalar cleanup loop would follow to execute the remainder of the instructions if the trip count n is not exactly divisible by four. Several steps are involved in this process. First, because it is possible that no information exists about the base addresses of the arrays, runtime code must be inserted to ensure that the arrays do not overlap (dynamic dependence testing) and that the bulk of the loop runs with each vector iteration having addresses aligned along 16-byte boundaries (dynamic loop peeling for alignment). In order to vectorize efficiently, only loops of sufficient size are vectorized. If the number of iterations is too small, a simple serial loop is used instead. Besides simple loops, the vectorizer also supports loops with reductions (such as summing an array of numbers or searching for the max or min in an array, conditional constructs, saturation arithmetic and other idioms. Even the vectorization of loops with trigonometric mathematical functions is supported by means of a vector math library.

To give a taste of a realistic performance improvement that can be obtained by intra-register vectorization, we report some performance numbers for the double-precision version of the Linpack benchmark (available in both Fortran and C at www.netlib.org/benchmark). This benchmark reports the performance of a linear equation solver that uses the routines DGEFA and DGESL for the factorization and solve phase, respectively. Most of the runtime of this benchmark results from repetitively calling the Level 1 BLAS routine DAXPY for different subcolumns of the coefficient matrix during factorization. Under generic optimizations (switch -O2), this benchmark reports 1,049 MFLOPS for solving a 100×100 system on a 2.66GHz Pentium 4 processor. When intra-register vectorization for the Pentium 4 processor is enabled (switch -xW), the performance goes up to 1,292 MFLOPS, boosting the performance by about 20%.

OpenMP and Auto-Parallelization

The OpenMP standard for C/C++ and Fortran (www.openmp.org) has recently emerged as the de facto standard for shared-memory parallel programming. It allows the user to specify parallelism without getting involved in the details of iteration partitioning, data sharing, thread scheduling and synchronization. Based on these directives, the Intel compiler will transform the code to generate multithreaded code automatically. The Intel compiler supports the OpenMP C++ 2.0 and OpenMP Fortran 2.0 standard directives for explicit parallelization. Applications can use these directives to increase performance on multiprocessor systems by exploiting both task and data parallelism.

The following is an example program, illustrating the use of OpenMP directives with the Intel C++ Linux OpenMP compiler:

```
#define N 10000
void   ploop(void)
{
  int k, x[N], y[N], z[N];
  #pragma omp parallel for private(k) shared(x,y,z)
```

```
     for (k=0;  k<N; k++) {
       x[k] = x[k] * y[k] + workunit(z[k]);
     }
   }
```

The for loop will be executed in parallel by a team of threads that divide the iterations in the loop body amongst themselves. Variable k is marked private—each thread will have its own copy of k—while the arrays x, y and z are shared among the threads.

The resulting multithreaded code is illustrated below. The Intel compiler generates OpenMP runtime library calls for thread creation and management, as well as synchronization (see Resources 1 and 2):

```
#define N 10000
void  ploop(void)
{
    int k, x[N], y[N], z[N];
    __kmpc_fork_call(loc,
                     3,
                     T-entry(_ploop_par_loop),
                     x, y, z)
    goto L1:
    T-entry _ploop_par_loop(loc, tid,
                            x[], y[], z[]) {
       lower_k = 0;
       upper_k = N;
       __kmpc_for_static_init(loc, tid, STATIC,
                              &lower_k,
                              &upper_k, ...);
       for (local_k=lower_k;  local_k<=upper_k;
            local_k++)  {
          x[local_k] = x[local_k] * y[local_k]
                     + workunit(z[local_k]);
       }
       __kmpc_for_static_fini(loc, tid);
       T-return;
    }
L1: return;
}
```

The multithreaded code generator inserts the thread invocation call __kmpc_fork_call with the T-entry point and data environment (for example, thread id tid) for each loop. This call into the Intel OpenMP runtime library forks a number of threads that execute the iterations of the loop in parallel.

The serial loops annotated with the OpenMP directive are converted to multithreaded code by localizing the lower- and upper-loop bounds and by privatizing the iteration variable. Finally, multithreading runtime initialization and synchronization code is generated for each T-region defined by a [T-entry, T-ret] pair. The call __kmpc_for_static_init computes the localized loop lower-bound, upper-bound and stride for each thread according to a scheduling policy. In this example, the generated code uses static scheduling. The library call __kmpc_for_static_fini informs the runtime system that the current thread has completed one loop chunk.

Rather than performing source-to-source transformations, as is done in other compilers such as OpenMP NanosCompiler and OdinMP, the Intel compiler performs these transformations internally. This allows tight integration of the OpenMP implementation with other advanced, high-level compiler optimizations for improved uniprocessor performance such as vectorization and loop transformations.

Besides the compiler support for exploiting the OpenMP directive-guided explicit parallelism, users also can try auto-parallelization by using the option -parallel. Under this option, the compiler automatically analyzes the loops in the program to detect those that have no loop-carried dependency and can be executed in parallel profitably. The auto-parallelization phase in the compiler relies on the advanced memory disambiguation techniques for its analysis, as well as the profiling information for its heuristics in deciding when to parallelize.

CPU-Dispatch

One of the unique features of the Intel compiler is CPU-Dispatch, which allows the user to target a single object for multiple IA-32 architectures by means of either manual CPU-Dispatch or Auto-CPU-Dispatch. Manual CPU-Dispatch allows the user to write multiple versions of a single function. Each function either is assigned a specific IA-32 architecture platform or is considered generic, meaning it can run on any IA-32 architecture. The Intel compiler generates code that dynamically determines on which architecture the code is running and accordingly chooses the particular version of the function that will actually execute. This runtime determination allows programmers to take advantage of architecture-specific optimizations, such as SSE and SSE2, without sacrificing flexibility, allowing execution of the same binary on architectures that do not support newer instructions.

Auto-CPU_Dispatch is similar but with the added benefit that the compiler automatically generates multiple versions of a given function. During compilation, the compiler decides which routines will gain from architecture-specific optimizations. These routines are then automatically duplicated to produce architecture-specific optimized versions, as well as generic versions. The benefit of this feature is, it does not require any rewrite by the programmer. A normal source file can take advantage of the Auto-CPU-Dispatch feature by the simple use of a command-line option. For example, given the function:

```
void init(float b[], double c[], int n)
{
  int i;
  for (i = 0; i < n; i++) {
      b[i] = (float)i;
  }
```

```
    for (i = 0; i < n; i++) {
        c[i] = (double)i;
    }
}
```

the Intel compiler can produce up to three versions of the function. A generic version of the function is generated that will run on any IA-32 processor. Another version would be tuned for the Pentium III processor by vectorizing the first loop with SSE instructions. A third version would be optimized for the Pentium 4 processor by vectorizing both loops to take advantage of SSE2 instructions.

The resulting function begins with dispatch code like this:

```
.L1  testl     $-512, __intel_cpu_indicator
     jne       init.J
     testl     $-128, __intel_cpu_indicator
     jne       init.H
     testl     $-1, __intel_cpu_indicator
     jne       init.A
     call      __intel_cpu_indicator_init
     jmp       .L1
```

Where init.A, init.H and init.J are the generic, SSE and SSE2 optimized versions, respectively.

Language Extensions

While the Intel compiler is strictly ANSI-compliant, there are options to cover many GCC extensions, such as long long int, zero-length arrays or macros with variable number of arguments. GCC-style inline assembly code is also supported. DWARF2 debugging information is provided to use with standard debuggers such as GDB. Certain Microsoft extensions are also enabled, such as __declspec attributes, along with support for Microsoft-style inline assembly code.

In addition to inline assembly code, the Intel compiler also supports MMX and SSE/SSE2 intrinsics. These allow access to the processor-specific extensions without the performance and correctness problems often caused by using inline assembly that can interfere with the analysis and transformations of the Intel compiler. By using the provided intrinsics, the programmer can take advantage of specific instructions but still receive the benefits of register allocation, scheduling and other optimizations.

Conclusions

The Intel compiler for Linux is a state-of-the-art compiler that delivers performance among the best in the industry, using sophisticated techniques to enable advanced features of Intel IA-32 architectures. More information can be found at developer.intel.com/software/products/compilers.

Acknowledgements

Thanks to Zia Ansari and David Kreitzer for their help in describing some of the technical details of the compiler. We also thank all the other members of the Intel compiler team.

Resources

Dale Schouten (Dale.A.Schouten@intel.com) works at the Intel Compiler Lab. He has a PhD from the University of Illinois. In his other life, Dale is an unprofessional musician and the father of two exceptional children.

Xinmin Tian (Xinmin.Tian@intel.com) works at the Intel Compiler Lab at Intel Corp. He manages the OpenMP Parallelization group. He holds BSc, MSc and PhD degrees in Computer Science from Tsinghua University.

Aart Bik (Aart.Bik@intel.com) received his MSc degree in Computer Science from Utrecht University, The Netherlands, and his PhD degree from Leiden University, The Netherlands. He is currently working on vectorization and parallelization at the Intel Compiler Lab.

Milind Girkar (Milind.Girkar@intel.com) received a PhD degree in Computer Science from the University of Illinois at Urbana-Champaign. Currently, he manages the IA-32 Compiler Development group at the Intel Compiler Lab.

Archive Index Issue Table of Contents

Advanced search

Large-Scale Mail with Postfix, OpenLDAP and Courier

Dave Dribin

Keith Garner

Issue #106, February 2003

Setting up an SMTP mail server for multiple domains on a single machine with remote access via IMAP.

Although this article provides instructions on setting up an integrated mail server using Postfix, OpenLDAP and Courier-IMAP, it does not discuss how these software components were chosen, which could be a whole article in and of itself. The goal is to set up an SMTP mail server for multiple domains on a single machine with remote access via IMAP. Also, instead of having mail delivered only to people with shell accounts, we want to have IMAP accounts that do not have a corresponding shell account. This gives rise to two classes of accounts: local and virtual. Local accounts are those with shell access. They use their shell user name and password to access IMAP. Virtual accounts have a user name and password that only works for logging in to IMAP. The terms local and virtual are used throughout the rest of the article.

The Big Picture

Figure 1 shows how Postfix, Courier, Procmail and OpenLDAP interact. Local account information is stored in /etc/password, and authentication is handled by pluggable authentication modules (PAM). Virtual account information is stored in an LDAP directory. LDAP provides both account lookup and authentication capabilities. It is possible to avoid an LDAP directory, but it will be more difficult to administer the virtual account information. For example, Postfix and Courier both support virtual accounts using configuration files, but they have different file formats.

Postfix accepts incoming mail from SMTP. It will reject any mail for unknown accounts, both local and virtual. It delivers the mail itself for virtual accounts and uses Procmail as the MDA for local accounts. Courier provides remote access to the mailboxes via the IMAP and POP protocols.

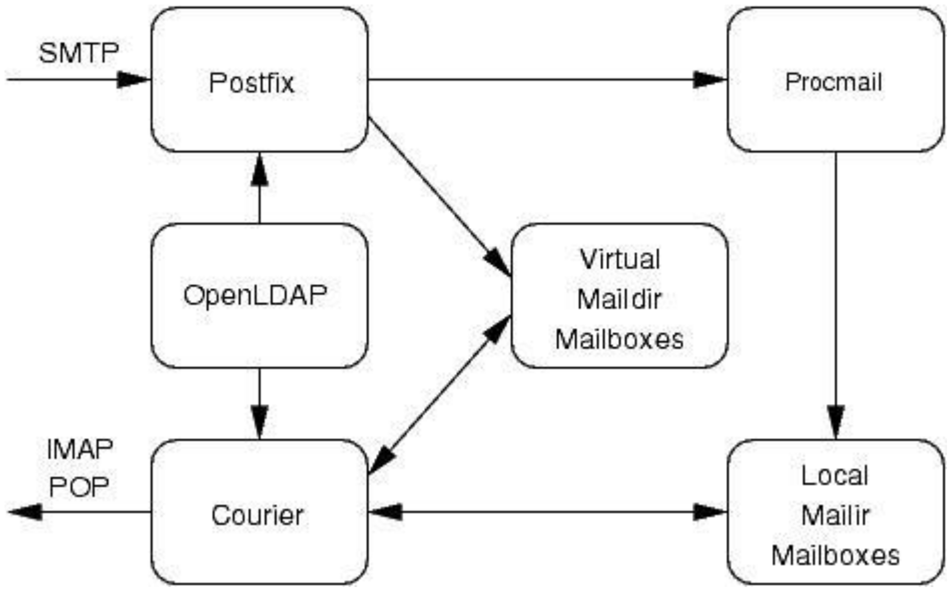


Figure 1. Overall Design

Mailbox Location

A local account's mail is stored in its home directory at ${HOME}/Maildir/ in the Maildir format. It is standard practice for Maildir delivery to go into the account's home directory rather than /var/spool/mail. Both Postfix and Courier work out of the box with this standard behavior.

Unlike local accounts, there is no standard location for virtual accounts' e-mail. We created a single UNIX account, called vmail, that holds the mail for all the virtual accounts. Each virtual domain has a subdirectory within the ~vmail/domains/ directory. For example, if there is an account <john@example.com>, mail would be stored in ~vmail/domains/example.com/john/ in maildir format. You can also spread virtual accounts across multiple UNIX accounts, for example, by creating a UNIX account for each virtual domain.

LDAP Directory Design

There are many possibilities when designing your directory, and not all aspects of this topic are covered here. One useful reference is the iPlanet Deployment Guide (see Resources). This article assumes you are familiar with LDAP concepts and terminology. You should take the time up front to design a tree that matches your specific requirements.

Tree Structure

Figure 2 shows a sample directory tree for a web hosting company. The company's domain name, myhosting.example, was chosen as the root suffix.

Postfix and Courier both search the o=hosting,dc=myhosting,dc=example subtree for e-mail information. The o=accounts,dc=myhosting,dc=example subtree shows how you could integrate shell account information for PAM into the same directory, but this is not necessary for setting up e-mail. Each hosted domain gets its own organization beneath the hosting organization. Each e-mail account goes under the domain's subtree. Thus, the distinguished name for the <user2@domain2.example> e-mail address is:

```
mail=<user2@domain2.example>,o=domain2.example,
 o=hosting,dc=myhosting,dc=example
```

This is a fairly stable design as accounts will never transfer between domains. The end result is good LDAP design, because moving subtrees can be troublesome in LDAP. The design is also quite flexible because each domain's tree can be tailored, if necessary. Each domain must have a postmaster entry that provides dual functionality. Its primary function is for access control, but it also acts as a forwarding e-mail address. Each domain also must have an abuse alias that forwards mail to the system administrator.

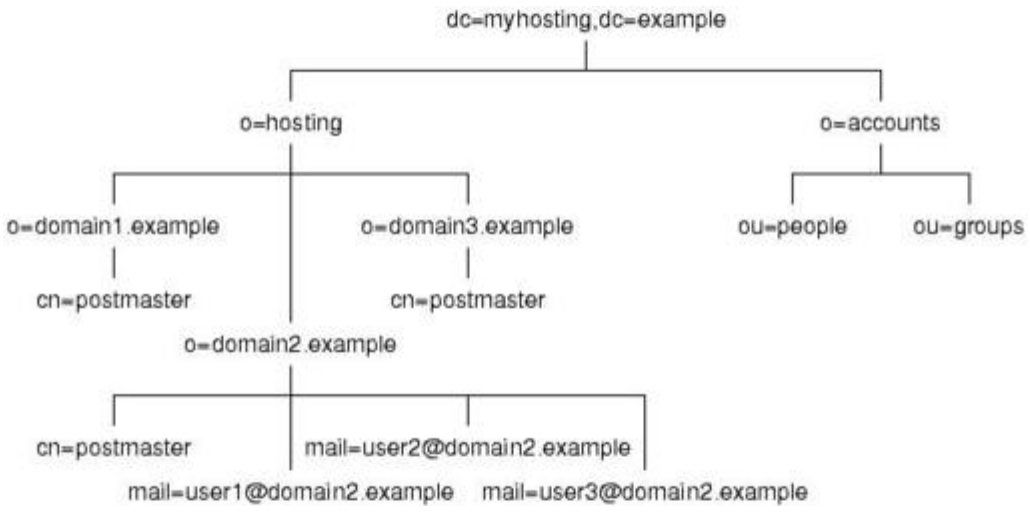


Figure 2. A Sample Directory Tree for a Web-Hosting Company

Choosing a Schema

The schema defines which attributes an entry can have by defining object classes. None of the default schemas that come with OpenLDAP are really suited for entries used exclusively for e-mail mailboxes or forwarding. We are using the schema that Courier provides in its distribution. Another possible schema to look at is the schema distributed with the qmail-LDAP Project. You also can design your own schema, but be aware that you should use OIDs registered with the Internet Assigned Numbers Authority (IANA).

Courier Schema

The courierMailAccount object class, summarized in Table 1, is used for virtual e-mail accounts. The courierMailAlias object class, summarized in Table 2, is used for e-mail addresses that forward to another address.

Table 1. courierMailAccount

Table 2. courierMailAlias

The courierMailAccount object class does not exactly fit our needs. We do not need uidNumber and gidNumber because all mail goes to the vmail account. However, we must put in dummy values as the schema requires them. Note that these values would be meaningful if we were spreading virtual accounts over many UNIX accounts. We require the mailbox attribute, because it is needed to determine the location of the mailbox on the filesystem. The mailbox must end in a slash to indicate that it's a Maildir-style mailbox. The userPassword attribute also is required because all e-mail accounts must have a password in order to be accessed via IMAP or POP. We do not use the other optional attributes.

The courierMailAlias object class is a good fit for our needs. We use only the two required attributes and do not use either of the optional attributes. The maildrop attribute can be another e-mail address or a local account on this machine.

Access Control

OpenLDAP provides many possibilities for access control. By default, the root account has read and write access to all entries in the tree. We would like to delegate some of this administration to individual accounts in each hosted domain so they can do minor changes on their own without access to the root account. This is done by making the postmaster entry an organizationalRole with a roleOccupant attribute for each entry with administration privileges. OpenLDAP can then be configured to allow access only to members of this group.

Implementation

This section describes how to implement a virtual mail solution. Not every little detail is covered, only what is needed above and beyond the standard installations.

Following is the list of software, with version numbers, with which we tested this configuration:

- Red Hat Linux 6.2, 7.1, or 7.2
- Postfix 1.1.x
- OpenLDAP 2.0.21
- Courier-IMAP 1.4.1
- Procmail 3.22

You need to create the vmail account, and then create the ~/vmail/domains/ directory. You also need to create an account and two groups for Postfix as covered in Postfix's INSTALL documentation.

You do not need to follow any special instructions for compiling and installing OpenLDAP, so consult its documentation for instructions. For a production environment, read up on running OpenLDAP as a non-root account, setting up a chroot environment and replication. This article describes how to configure slapd for a single server, create the base tree structure and insert some basic data into the LDAP directory. Figure 2 shows the LDAP tree we set up here.

Configuring slapd

You need to make Courier's schema file available, so copy the file from authlib/authldap.schema in the Courier distribution to /usr/local/etc/openldap/schema/courier.schema. Courier's schema depends on cosine.schema and nis.schema. Add these lines to slapd.conf:

```
include   /usr/local/etc/openldap/schema/cosine.schema
include   /usr/local/etc/openldap/schema/nis.schema
include   /usr/local/etc/openldap/schema/courier.schema
```

Next, set up a database definition with the following lines in slapd.conf:

```
database      ldbm
directory     /usr/local/var/openldap-ldbm
suffix        "dc=myhosting,dc=example"
```

The database directive specifies the back-end type to use (use LDBM as the back-end database). The directory directive specifies the path to the LDBM database. Make sure the directory specified exists prior to starting slapd and that slapd has read and write permissions on the directory. The suffix directive specifies the root suffix for this database. The next few lines set up the superuser or root account:

```
rootdn       "cn=Manager,dc=myhosting,dc=example"
rootpw       {SSHA}ra0sD47QP32ASAlaAhF8kgi+8Aflbgr7
```

The rootdn entry has complete access to the database, which is why the password is stored outside the actual database. The password in rootpw should always be stored in hashed format. Do not store the password in clear text. To convert the clear text password secret to a hashed format, use the slappasswd command:

```
% slappasswd
New password: secret
Re-enter new password: secret
{SSHA}ra0sD47QP32ASAlaAhF8kgi+8Aflbgr7
```

Take the output from slappasswd, and copy that into slapd.conf, as we did above.

To speed up searches, you should create indexes for commonly searched attributes:

```
index   objectClass  pres,eq
index   mail,cn      eq,sub
```

The last part in slapd.conf is the access control. The OpenLDAP FAQ contains good information on how you would set up postmaster as a group ACL.

Creating the Directory Tree

Now that slapd is configured, it's time to start adding data to the LDAP directory. We use the command-line tools that come with OpenLDAP and create LDIF files to modify the directory.

The first step is to create a base tree structure with our root node, the hosting organization and an entry for the rootdn. Create a file called base.ldif with the following contents:

```
dn: dc=myhosting, dc=example
 objectClass: top
 dn: cn=Manager, dc=myhosting, dc=example
 objectClass: top
 objectClass: organizationalRole
 cn: Manager
 dn: o=hosting, dc=myhosting, dc=example
 objectClass: top
 objectClass: organization
 o: hosting
```

Now use ldapadd, binding as the root account, to add this LDIF:

```
ldapadd -x -D "cn=Manager,dc=myhosting,dc=example" \
-w secret -f base.ldif
```

Adding a Domain

Domains can now be added under the hosting tree. Each domain needs to have postmaster and abuse entries at minimum. To make a tree for domain1.example, create a file called domain1.example.ldif with the following contents:

```
dn: o=domain1.example, o=hosting, dc=myhosting,
 dc=example
objectClass: top
objectClass: organization
o: domain1.example
dn: cn=postmaster, o=domain1.example, o=hosting,
 dc=myhosting, dc=example
objectClass: top
objectClass: organizationalRole
objectClass: CourierMailAlias
cn: postmaster
mail:
maildrop: postmaster
dn: mail=abuse@domain1.example, o=domain1.example,
 o=hosting, dc=myhosting, dc=example
objectClass: top
objectClass: CourierMailAlias
mail:
maildrop: abuse
```

Notice that the maildrop attributes are local e-mail accounts and will forward to the postmaster and abuse accounts in /etc/aliases. There are no accounts in the postmaster role, so only the root account can create accounts at the moment. Add this domain with the following command:

```
ldapadd -x -D "cn=Manager,dc=myhosting,dc=example"
\
-w secret  -f domain1.example.ldif
```

Adding an Account

Now, let's add an account with an e-mail <user1@domain1.example>. Let's also grant this account postmaster privileges for domain1.example. Create a user1.domain1.example.ldif with the following contents:

```
dn: mail=user1@domain1.example, o=domain1.example,
 o=hosting, dc=myhosting, dc=example
objectClass: top
objectClass: CourierMailAccount
mail:
homeDirectory: /home/vmail/domains
uidNumber: 101
gidNumber: 101
mailbox: domain1.example/user1
dn: cn=postmaster, o=domain1.example, o=hosting,
dc=myhosting, dc=example
changetype: modify
add: roleOccupant
roleOccupant: mail=user1@domain1.example,
 o=domain1.example, o=hosting,
 dc=myhosting, dc=example
```

The first section adds a new entry for the account. The home directory and mailbox point to the physical mailbox on the filesystem. The uidNumber and gidNumber attributes are required but not used, so they are filled in with dummy values of 101. The second section modifies the postmaster entry by adding a roleOccupant attribute with the DN of user1@domain1.example. Let's create this account:

```
ldapadd -x -D "cn=Manager,dc=myhosting,dc=example"
\
-w secret -f user1.domain1.example.ldif
```

The account does not have a password yet, so even though it has been granted postmaster privileges, it cannot be authenticated. Use the ldappasswd command to set the initial password to user1:

```
ldappasswd -x -D "$DN" -w $PW -s user1 \
"mail=user1@domain1.example, o=domain1.example,
o=hosting, dc=myhosting, dc=example"
```

Other domains and accounts can be added with similar LDIF files. Creating LDIF files by hand can be cumbersome and error-prone. We discuss alternatives for administration later.

Postfix

We cover only the sections of Postfix that pertain to the mail hosting. To deal with other parts of Postfix setup, please visit the Postfix web page.

Download the Postfix source and untar it. You need to rebuild the Postfix Makefiles to be aware of LDAP and link against it. To do this, execute the following command:

```
make makefiles CCARGS="-I/usr/local/include
-DHAS_LDAP" AUXLIBS="-L/usr/local/lib -lldap
-L/usr/local/lib -llber"
```

At this point, follow the normal Postfix compiling and installing instructions as documented in its INSTALL and LDAP_README files.

Configuring Postfix

While configuring Postfix for this task, we are mostly concerned with /etc/postfix/main.cf. For most of the Postfix configuration, you will configure in a way that makes the most sense for your site, and you can follow the documentation contained in the Postfix source or on the Postfix web site. Here, we talk about the settings that are affected by this setup. If any of the configuration examples shown below aren't explicitly attributed to a specific file, assume they can be found in main.cf.

The transport table maps domains to message delivery transports (as specified in /etc/postfix/master.cf) and/or relay hosts. For our virtual domains, we want to map them to the virtual delivery agent that comes with Postfix. A transport table could look something like this:

```
domain1.example             virtual:
domain2.example             virtual:
```

After making your transport table in plain text, you need to make it into a binary DB file using postmap (see **man postmap**). At this point, tell Postfix that there is a transport table and where to find it. You also need to let Postfix know that we accept mail for those domains. This is done through the transport_maps and mydestination directives:

```
transport_maps = hash:/etc/postfix/transport
mydestination = $myhostname, localhost.$mydomain,
  $mydomain, $transport_maps
```

You can define multiple LDAP sources easily. LDAP source parameters are documented in README_FILES/LDAP_README in the Postfix source. The parameter names follow the pattern of <ldapsource>_parameter. The LDAP source name is defined by use. In main.cf, you'll need one LDAP source definition per each lookup.

Aliases

The first LDAP source definition is for virtual aliases. We've named this LDAP source aliases. In our configuration, our LDAP server is running on localhost. The search base is the top of the hosting subtree we defined in our LDAP server. We're querying for items where the mail elements match the e-mail recipient as well as items that are of the courierMailAlias object class. The destination of the alias is stored in the maildrop attribute. Postfix won't bind using an account, instead it will do an anonymous lookup:

```
aliases_server_host = localhost
aliases_search_base =
  o=hosting,dc=myhosting,dc=example
aliases_query_filter =
  (&(mail=%s)(objectClass=CourierMailAlias))
aliases_result_attribute = maildrop
aliases_bind = no
```

Accounts

When using the accounts source we're looking for entries that have an object class of courierMailAccount. We request the mailbox attribute as the result:

```
accounts_server_host = localhost
accounts_search_base =
  o=hosting,dc=myhosting,dc=example
accounts_query_filter =
  (&(mail=%s)(objectClass=CourierMailAccount))
accounts_result_attribute = mailbox
accounts_bind = no
```

A second source for accounts, accountsmap, also needs to be defined to help locate accounts when a catchall is used. Without this lookup, a catchall in the aliases would override virtual accounts in a domain:

```
accountsmap_server_host = localhost
accountsmap_search_base = o=hosting,dc=myhosting,dc=example
```

```
accountsmap_query_filter =
(&(mail=%s)(objectClass=CourierMailAccount
accountsmap_result_attribute = mail
accountsmap_bind = no
```

Now that the aliases and accountsmap LDAP source are defined, let Postfix know to use it by defining the virtual_maps parameter in main.cf:

```
virtual_maps = ldap:aliases
```

For this example, assume there is a vmail UNIX account created that has a UID of 125, a GID of 120 and its home directory is /home/vmail:

```
:virtual_mailbox_base = /home/vmail/domains
virtual_mailbox_maps = ldap:accounts
virtual_minimum_uid = 125
virtual_uid_maps = static:125
virtual_gid_maps = static:120
```

Set the virtual_uid_maps and virtual_gid_maps to a special static map and hard code it to the UID and GID of the vmail account. All of the parameters shown here are fully documented in README_FILES/VIRTUAL_README, which comes with the Postfix source.

We also need to edit the local_recipient_maps parameter to look at the virtual_mailbox_maps so Postfix knows what accounts are valid. This is needed so Postfix can reject mail for unknown accounts:

```
local_recipient_maps = $alias_maps
  unix:passwd.byname $virtual_mailbox_maps
```

Courier

There aren't any special instructions for installing Courier, so see its documentation for full instructions. It should autodetect LDAP and build it in. You should seriously consider passing the --enable-workarounds-for-imap-client-bugs option to ./configure, otherwise Netscape mail users may have trouble interacting with your server. This bends the IMAP protocol a little bit, but it's better to have happy users than a perfect protocol with unhappy users.

Courier uses an authentication dæmon to keep authentication separate from the other parts of the system. Configure it so that a valid e-mail account is either found in either LDAP or PAM. Specify this in authdaemonrc using the authmodulelist parameter:

```
authmodulelist="authldap authpam"
```

All LDAP parameters are in authldaprc. Most parameters are self-explanatory. To use the Courier schema, you actually have a few modifications to make, though. You also need to map all virtual accounts to the vmail account. Here is a summary of the updates you need to make to authldaprc:

```
LDAP_GLOB_UID           vmail
LDAP_GLOB_GID           vmail
LDAP_HOMEDIR            homeDirectory
LDAP_MAILDIR            mailbox
LDAP_CRYPTPW            userPassword
```

Three other settings to be concerned with are LDAP_AUTHBIND, LDAP_BINDDN and LDAP_BINDPW. These relate to authenticating the user. LDAP_AUTHBIND is mutually exclusive with LDAP_BINDDN and LDAP_BINDPW. We recommend using LDAP_AUTHBIND. A comment in authldaprc mentions a memory leak in OpenLDAP when using LDAP_AUTHBIND, but it has been fixed in OpenLDAP version 2.0.19.

If you use LDAP_BINDDN and LDAP_BINDPW, you are limited to the crypt, MD5 and SHA algorithms for passwords. SMD5 and SSHA are not available. Also, you must put the root LDAP password in clear text in authldaprc when defining LDAP_BINDPW. There are security issues with putting the root LDAP password in clear text, so definitely use LDAP_AUTHBIND if you can.

The last change is to enable the IMAP server by setting the IMAPDSTART parameter to YES. You should now be able to use the courier-imap.sysvinit startup script to start and stop the IMAP dæmon.

Administration

Most of the administration tasks, such as adding, modifying and deleting accounts and aliases, require modifying the LDAP directory. You can do this with the OpenLDAP command-line tools or a generic LDAP browser like gq. These methods are cumbersome, however, because they are generic tools and are not tailored to the task of administering e-mail accounts. We've been working on a web administration application called Jamm that is essentially an application-specific LDAP browser written in Java and JSP. It also has its own LDAP schema that is a slightly modified Courier schema. Jamm is currently usable and is constantly evolving. Visit the Jamm web page on SourceForge for the latest Jamm information.

Account Creation Notes

When you create an account or an alias inside the LDAP database it will instantly become active as far as the mail system is concerned. For virtual accounts, note that the UNIX directory in ~vmail is not created at this time. However, we can work around this because Postfix's virtual delivery agent will create the necessary directories the first time it has to deliver mail. Due to this fact, we recommend sending a welcome e-mail as soon as you create the account.

Account Deletion Notes

When you delete an account or an alias in the LDAP database, it will instantly become inactive. For virtual accounts, note that the UNIX filesystem isn't cleaned up. In other words, the data remains on disk until a system administrator can remove it. This allows you to keep the data from dead accounts for a grace period in case the account was deleted in error. However, if another account is created with the same name and the same mail path, the data will be available to the new account. This could be considered a privacy violation for the previous user.

Resources

Dave Dribin (dave@dribin.org) has been using UNIX since 1991 and Linux since 1993. He has been professionally developing software for or on UNIX since 1995. Dave is currently working as an independant consultant at the National Association of Realtors.

Keith Garner has been using Linux since January 1994. He has been professionally administrating and developing software for UNIX since 1997. Keith is currently employed by the National Association of Realtors.

Archive Index Issue Table of Contents

Advanced search

Linux from Kindergarten to High School

Michael Surran

Issue #106, February 2003

Moving the school computer lab to Linux was not an easy decision to make—but it was a beneficial one.

As the bell rings to begin class at Greater Houlton Christian Academy, enthusiastic students sit down at their shiny, new computer workstations. In one corner, the red cabinet housing the server hums quietly as two stuffed penguins look on fondly from their perch. Other penguins keep watch from different locations as the students enter their user names and passwords to access their accounts. Ask a student who “Tux” is, and he or she will point to the large penguin painted on the front wall of the computer lab and say, “He's the Linux penguin!” About this time KDE has loaded, and young boys and girls are opening the application they need for class as easily as kicking a ball.

Figure 1. First graders learning some penguin art fundamentals.

Now for a little history. Greater Houlton Christian Academy (GHCA) is a private school and nonprofit organization in Maine. As such, it does not have the same access to funding as the public school system. As the computer science teacher and system administrator, this means I have to be creative about providing our students with computer technology while working with a tight budget. In the past I relied on area businesses and generous individuals to donate their used computers. While these donations were a great blessing to us, they were a temporary solution at best.

Last year it became quite evident that we would need to replace our old, secondhand computers running Windows 95. The decision to move from donated computers to new computers was based on many factors, though our primary goal was to make sure our students had the best technology available for the enhancement of their educational experience. Therefore, this would be a software upgrade as well as a hardware upgrade. In fact, choosing the software was by far the bigger challenge.

Interestingly enough, it was during this time that many schools in the western US were being audited by Microsoft concerning the school's use of Windows and Office software. I began to realize my ignorance concerning exactly how strict and inflexible the Microsoft EULA is. It was also during this time that Microsoft's new licensing initiative, called Software Assurance, was causing quite a stir in the tech headlines. As my research opened my eyes to the various limitations to proprietary software, I began to think that the answer for us might be found in open-source software.

The decision to switch to an open-source platform for our new computer lab was not an easy one. My experience was with DOS and various versions of Windows and not with UNIX-compatible operating systems. I had experimented with Linux a few years earlier but found it somewhat difficult and incomplete. Because some time had passed, I decided to give Linux another try. Going with Mandrake's 8.0 distribution, I installed Linux at home to see if it could replace Windows in a desktop environment. To my amazement, I found Linux to be much more capable this time around. I was one step closer to making my decision to switch our computer lab to the Linux OS.

Other factors went into the final decision to go with open-source software, not the least of which was cost. By purchasing bare-bones computer “kits”, we were able to save considerable money on the hardware. Part of the savings in purchasing a bare-bones system is that the computer does not come with an operating system. We knew by then we would have to spend more money on software than we did on hardware if we went with Microsoft. Not only would I need to consider the initial purchase of the operating system and application software, but I would also need to factor in the costs of upgrading our software every couple of years. Needless to say, going with an open-source platform would save us considerable money now and in the future.

Another key issue was flexibility. As many of you know, it takes time to install an operating system, customize it for the particular hardware it runs on and install the desired applications. Having purchased 20 new, identical computers, it made sense to completely configure one machine and then clone the hard drive to the other 19 computers. However, Microsoft's EULA prevents a user from doing this, even if they have 20 copies of Windows. Not only would Linux save me considerable time by allowing me to clone my configured PC, it also gave me great flexibility in the degree to which I could customize the OS for the hardware. By recompiling the kernel to take advantage of our specific hardware, I could fine-tune the OS to run at peak performance. Linux would even save us money in the cloning process, thanks to the dd command.

A few aspects, however, made the decision to switch to Linux a difficult one. The smaller software base to choose from and the lack of mature drivers for our hardware were among the lesser obstacles. The major obstacle was my own lack of experience with the Linux OS. In fact, most of the money and time spent in the software upgrade of our computer lab was for a shelf full of books I had to purchase and read to really feel confident using and teaching Linux. It isn't always easy to teach an old dog new tricks, but I found the experience one of the most challenging and rewarding experiences of my IT career.

Today our private school of over 170 students has one of the finest computer labs in Maine. We have 20 computers with Athlon 1600+ XP processors, 128MB of RAM, 20GB hard drives and all the accessories—3-D graphics, sound, 17" monitors and 100Mbps Ethernet networking. Our computers run Mandrake Linux 8.2 with KDE 3.0.2. What is most amazing is we upgraded our computer lab for under half the cost of what many neighboring schools paid for inferior equipment. Most of this savings was the result of switching to Linux.

Our servers also run Linux. Using NFS, students can access their accounts from any computer in the lab. Student- and staff-owned files are backed up on a daily basis, so gone are the days of “the computer lost my homework.” Our proxy server runs Squid to help speed our wireless internet connection to 20 workstations, and we use proxy software along with iptables to provide firewall protection. A nice program called Dansguardian provides filtering to protect our children from pornography and other inappropriate content.

Many of you may be asking at this point, “How do you use Linux in teaching your students?” GHCA is a K-12 school, and so we strive to offer some level of computer training for each grade. Kindergarten students, for example, can use such programs as Potato Guy to practice hand-eye coordination and familiarize themselves with how to use a mouse to manipulate objects on the computer screen. Elementary and secondary teachers integrate the computer lab into their curriculum by using the computer for research, multimedia enhancements or even something simple as coloring digital pictures.

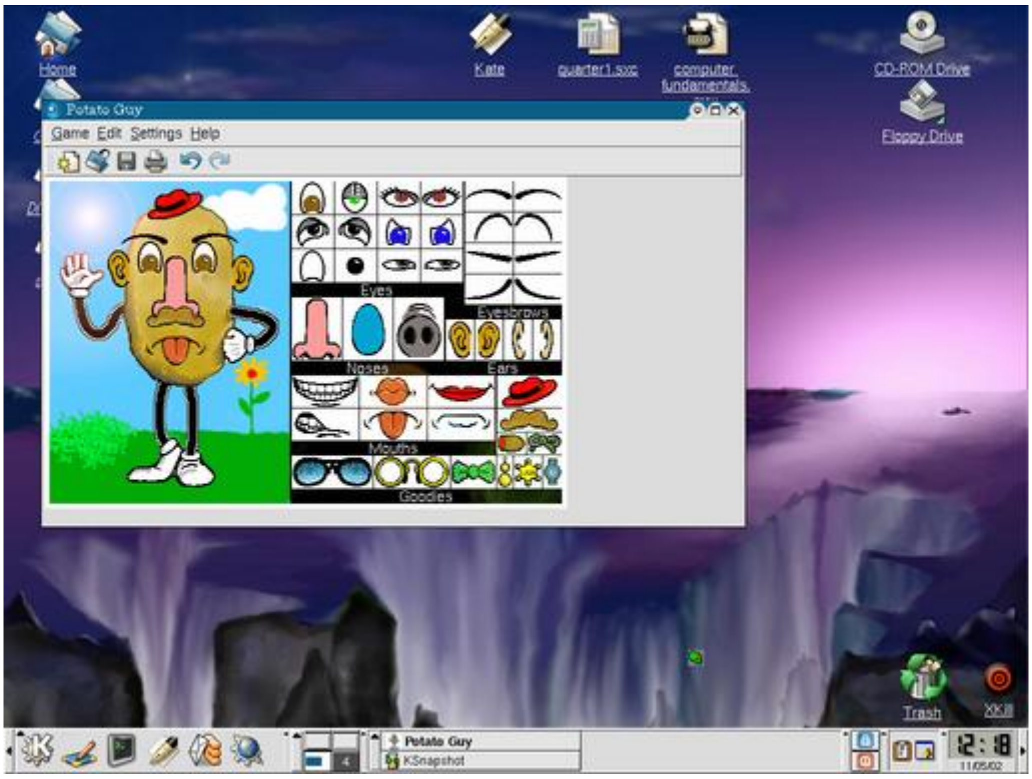
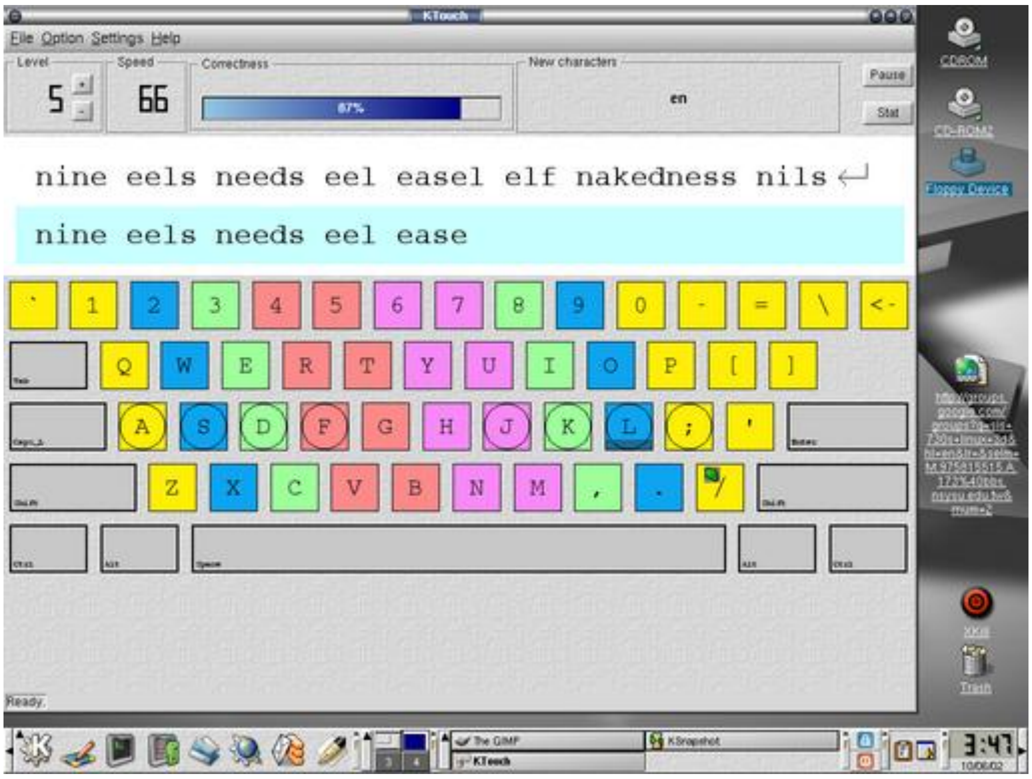


Figure 2. Potato Guy develops mouse skills.

Starting with grade seven, education in computer science takes a more formal approach. Seventh graders are taught keyboarding skills using programs such as KTouch and TuxTyping. Grade-eight students are taught the basics of programming with the kate editor and yabasic interpreter. It is during this class that students gain a better understanding of how computers process instructions.



nine eels needs eel easel elf nakedness nils

nine eels needs eel ease

Figure 3. Students learn touch typing with KTouch.

Computer Fundamentals is a one-credit course that introduces the ninth-grade student to “how a computer works” and “how to work a computer”. During the second semester, students learn about the purpose and use of the operating system and various applications, such as word processors, spreadsheets and web browsers. Because our computers run Linux, it is the Linux OS and open-source software that students learn in this class. Being sensitive to the fact that Microsoft currently dominates the PC market in corporate America, I do spend time discussing the similarities and differences between Linux and Windows.

Tenth- through twelfth-grade students can chose from a variety of computer electives, including how to upgrade and repair computers, web site design, advanced programming and even an upcoming course in robotics. In making the switch to Linux, I easily found all the tools needed to teach these courses using open-source software. In many cases, the open-source software we now use is superior to the proprietary software originally donated to us.

This is our first year with our new computer lab, and I am very pleased with how it is progressing. One of the most pleasing experiences I am having as a system administrator of a Linux-based lab is the actual ease of administration. Once I set something up in Linux, I rarely need to worry about it again. This was not the case with Windows. Last year we were constantly suffering from system crashes, frozen servers, strange bugs and the infamous “blue screen of death”. Needless to say, it was a frustrating situation for many students. While Linux is not bug-free, it has been a far more stable operating system for both our workstations and servers. Linux also has shown itself to be a much more versatile operating system to administer in a network environment. My job is more pleasurable thanks to our switch to Linux.

As a teacher of computer science, I am finding this year a fascinating test for Linux. Very few of our students, parents or teachers knew what Linux was before this year. I have actually found this to be a great advantage in teaching computers. In the past, I have found students to be disinterested in learning about the personal computer running Windows, because it is something most of them grew up with at home. This lack of interest made it more difficult to teach the more-advanced aspects of the operating system. However, Linux is something completely new, different and unexplored. Instead of being intimidated by the change, as many adults might be, young people are excited to explore the “uncharted territory”. This opens a door for me as a teacher, allowing me to educate eager minds in the more-advanced aspects of computer operating systems and software. In fact, it only took two weeks until students began to ask me, “Where can I get Linux?”

People sometimes ask me, “Is teaching our students Linux preparing them for the workplace?” This question is based on the fact that Microsoft is the current dominating presence in operating systems and office software. It is a question I have thought over a long time, and the answer I always come up with is, “Yes, most definitely.” The basic principles of any type of operating system, office application or other similarly grouped software are the same. A student who becomes proficient in Linux will not find themselves lost in a Windows environment. I have found Linux to be the more advanced of the two operating systems, yet our students are very quickly and easily learning it. The process of copying a file or formatting a paragraph is not so different between one operating system and the other. The important thing is we are able to offer the latest in hardware and software tools to train our students in these fundamental principles—something we could not do if we went with proprietary software.

Another question that may be even more important to ask is, “What is the future of Linux?” When our students graduate a few years from now, will they enter a Microsoft-dominated workplace or will the tide have changed? Even in our small New England town of Houlton, Maine, businesses are beginning to look to Linux as an alternative to proprietary operating systems. These businesses will need qualified personnel familiar with the Linux operating system and open-source applications. Greater Houlton Christian Academy will be graduating young men and women who will be able to meet that need, a claim not many schools in our nation can currently make. In fact, some of our students may go on to write the future applications for Linux, giving back to the community that helped them during their school years.

For us, switching to open-source software running on the Linux operating system has been the right choice, allowing us to provide our students with modern equipment and software for a fraction of the cost of a computer lab running proprietary software. If Linux continues to grow in popularity and gain a foothold in the workplace, we will look back at our choice as one of the most important decisions we've ever made.

email: computerlab@ghca.com

Michael Surran is the system administrator and computer science teacher at Greater Houlton Christian Academy (www.ghca.com) in Northern Maine. Michael enjoys church, outdoor adventures, target shooting, sci-fi, collecting penguins and his wife, Lisa, who also teaches at GHCA.

Archive Index Issue Table of Contents

Advanced search

Removing Red-Eye with The GIMP

Eric Jeschke

Issue #106, February 2003

Tired of photos where all your family and friends look like Satan's spawn?

With the abundance of low-cost digital cameras and scanners available these days, more and more Linux users are looking to the popular, open-source GNU Image Manipulation Program (The GIMP) when editing their digital images. This article describes a simple technique for eliminating the dreaded “red-eye” from your candid flash photos using The GIMP.

First, a little background on red-eye—what causes it and what you can or can't do to avoid it. You can see a good example of red-eye in Figure 1. The little girl's red eyes are not the product of staying up late and reading *LJ*; it's a result of an ordinary flash from a digicam. It's important to note that red-eye occurs only under a particular combination of circumstances: the camera is within a particular range of the subject, the subject is looking toward the camera and the flash is close to the taking angle of the lens. The problem is caused by the intense directed light of the flash reflecting off the retina in the back of the eye, straight back into the camera lens. The color is red because the light is filtered through the blood capillaries in the eyes. For a more detailed explanation, see the one offered in the “How Stuff Works” series (www.howstuffworks.com/question51.htm).

If you must take a flash photo because of dim lighting, you can do several things at picture time to reduce or eliminate red-eye and avoid having to edit the image later.

- Move the flash farther off axis from the picture-taking angle, so the reflected angle is changed (e.g., using a flash bracket or remote trigger).
- Bounce the flash (e.g., off of a white ceiling).
- Move closer or farther away from the subject.

While all of these are effective at reducing or eliminating the red-eye effect, they are, unfortunately, not viable options for the casual shooter. For one thing, the flash on most consumer cameras is not designed to allow bounce flash and is fixed very close to the lens because the camera is designed to be compact. Furthermore, changing distance is not always an option because of the low power of the built-in flash unit, the constraints of the photographer's position, the lens' zoom limitations or other issues.

Most cameras sold in the last several years have a “red-eye reduction” flash mode. Why not just use that? The red-eye flash mode works in most cases by firing the flash repeatedly at one or more low-power settings before opening the shutter and firing the final full-power flash. The preflashes cause the subject's pupils to close down, thus providing a narrow exit channel for the reflected light, thus “reducing” red-eye. Unfortunately, it often also causes the subjects to grimace as their eyes, adjusted to a darkened setting, respond to the sudden intrusion of intense light. As a result, in many shots taken with red-eye reduction mode flash, the subjects have a “deer caught in the headlights” look, have their eyes closed or are squinting. So it seems impossible in many situations to avoid red-eye if you want a decent candid flash photograph. Here is where The GIMP comes to the rescue.

The technique I describe allows you to retain as much of the all-important *tonality* (shades of lightness and darkness) of the different areas of the pupil. Also, it preserves the *catchlight* (the reflection of light off of the cornea over the pupil that gives the impression of life and vitality in the subject).

Most of the menus in The GIMP are accessed by clicking the rightmost mouse button in an image window. In the description that follows, a right-click is abbreviated RC. If I describe a GIMP action that needs to be invoked, I mention the series of menus or a keyboard shortcut in parentheses. For example, open the image (RC-->File-->Open), means right-click in the image window and choose File and then from that menu choose Open. If a keyboard shortcut makes more sense, I'll list the combination of keys to press. For instance, copy the image (Ctrl-C) means press and hold the Control key and press C.

Here's the technique. Start The GIMP and load your red-eye image (RC-->File-->Open), as shown in Figure 1. Once you've got your image loaded, zoom in close on the eyes (by pressing + a couple of times and scrolling as necessary) so you can get a good, large view of the red pupils, as shown in Figure 2.

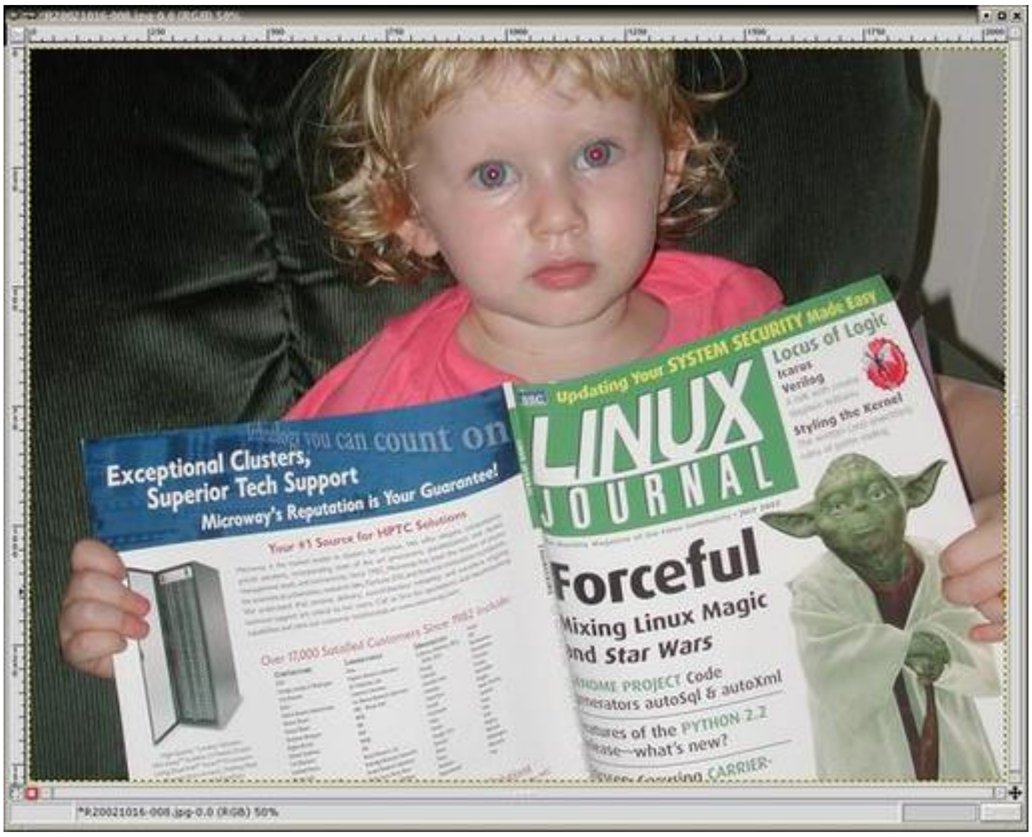


Figure 1. Sweet Little Girl—Or Devil Child?

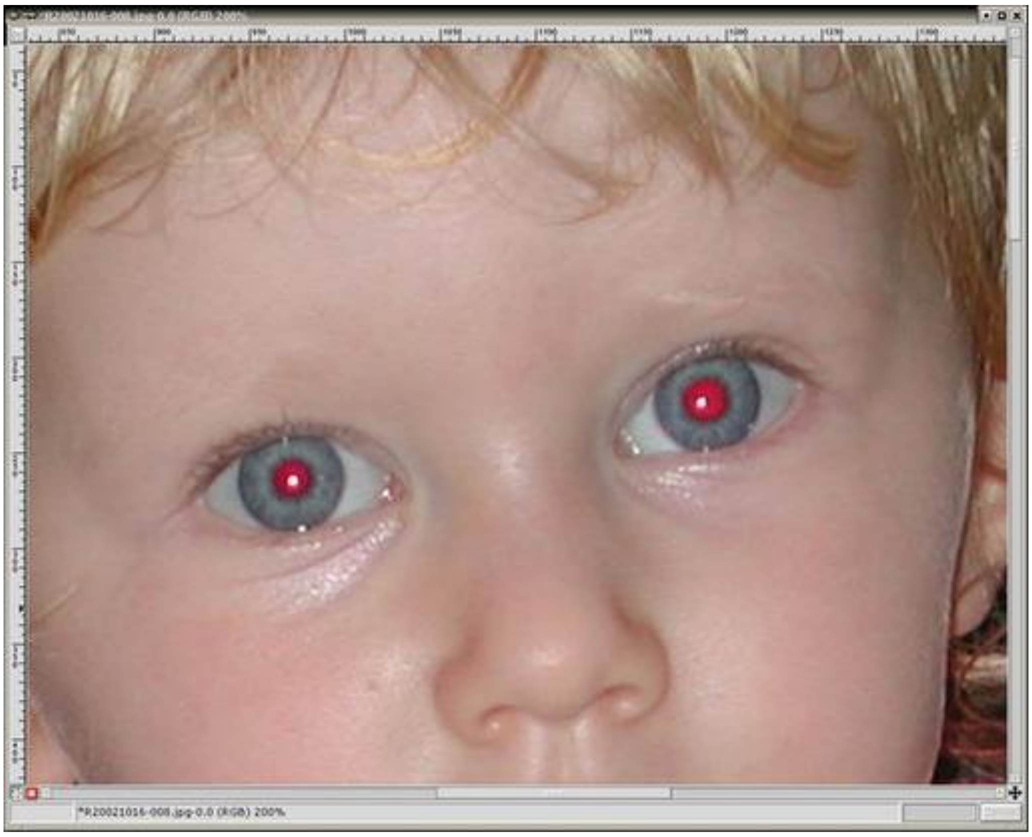


Figure 2. Close-up of the Pupils to Be Fixed

Next, we want to select only the pixels that constitute the pupils of the subject's eyes. There are a number of ways to go about this, but one I've found that seems to work fairly well is the fuzzy select tool, also known as the “magic wand”. The fuzzy select tool works by selecting contiguous areas of color/tone. By this we mean areas that differ by no more than a certain threshold in actual pixel values (that's where the “fuzzy” part comes in; we can control the threshold).

The fuzzy selection tool works best in areas of the image where contrast can be maximized between the desired selection and the rest of the image (in this case the pupil and the iris). All images in RGB mode (the normal mode for most color images) are made of three channels: red, green and blue, each containing pixel values for the entire image. If you examine these channels individually, you will usually find that the green channel has the most contrast for fuzzy selection of the pupils.

Bring up the Layers dialog (Ctrl-L) and click on the Channels tab. Then deselect the Red and Blue channels. The Layers dialog should look like Figure 3, with only the Green channel highlighted. We haven't turned off the visibility of these two channels, so the image window won't look any different. But by selecting only the green channel, our fuzzy select examines only the green channel pixel values in deciding which adjacent pixels to select.

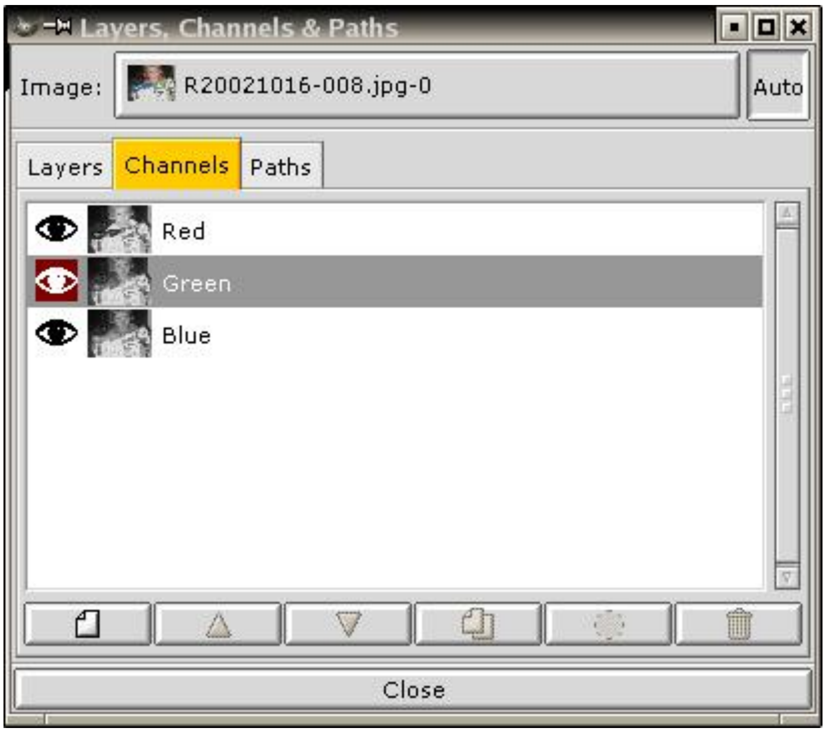
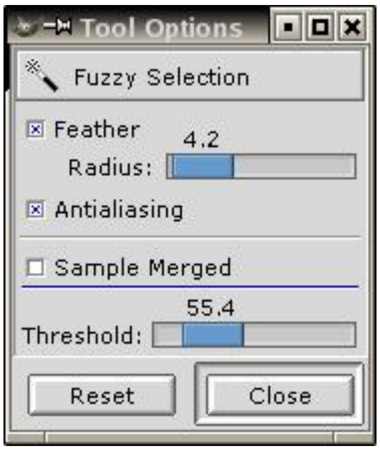


Figure 3. Color Channels in the Layer Display

Now double-click on the fuzzy select (magic wand) tool in The GIMP toolbox window to see the tool options. You'll need to experiment with the Threshold setting, but generally you'll want to increase it from the default. Try the value I used here, shown in Figure 4. You also should check the Feather option and give it a small amount, as shown.


Tool Options

Fuzzy Selection

[✓] Feather Radius: 4.2

[✓] Antialiasing

[ ] Sample Merged

Threshold: 55.4

Figure 4. Increasing the Threshold

Now click on the red part of one pupil in the image. You should see it select most of the pupil with a “marching ants” outline. If it doesn't, clear the selection (Shift-Ctrl-A), increase the Threshold value slightly and try again. Conversely, if areas outside the pupil are selected, clear the selection, reduce the Threshold and try again. Another possibility is to use the Grow Selection (RC-->Select-->Grow) and Shrink Selection (RC-->Select-->Shrink) dialogs to slightly enlarge or decrease the selection if it looks mostly right, but you are a few pixels off either way.

Once you've got a decent selection on one pupil, hold down the Shift key and click on the red part of the other pupil (holding Shift during a selection adds to the current selection). At this point you should have both pupils selected, as shown in Figure 5.

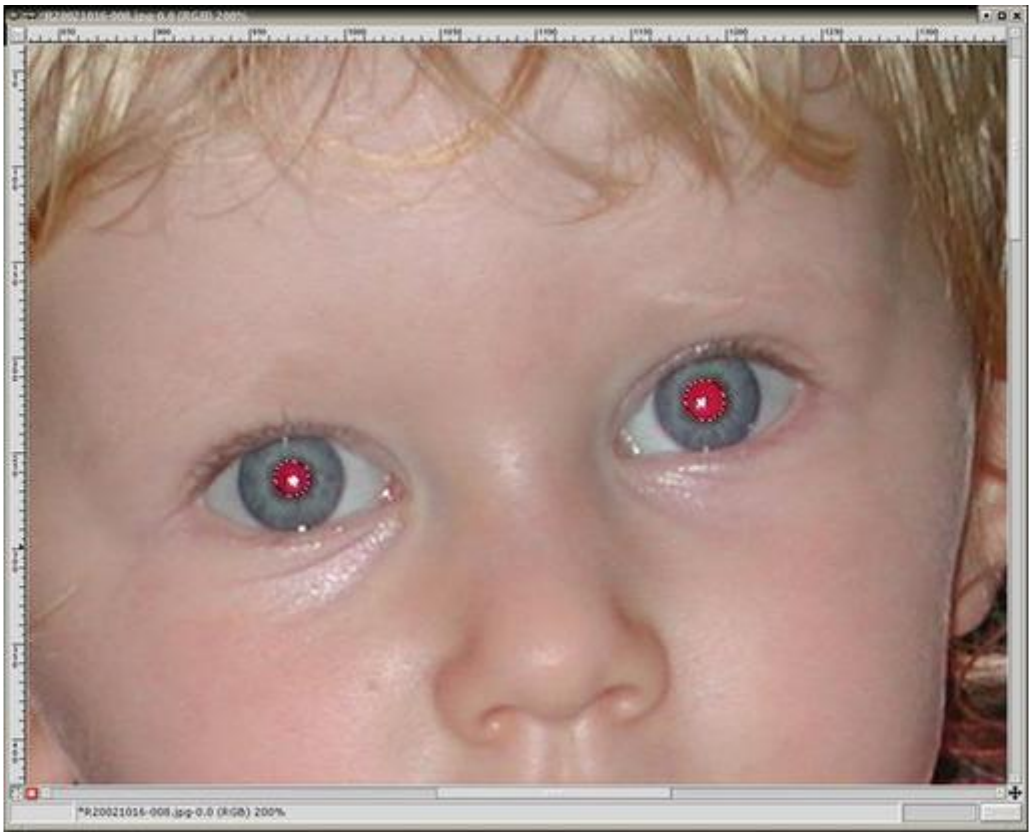


Figure 5. Both Pupils Selected

Tip for advanced GIMP users: if you know about quick masks, you can fix up an imperfect selection here. Click on the quick mask button, apply a few appropriate paint strokes with a small, fuzzy brush, and then go back to selection mode.

Now go back to the Layers dialog, select the Red channel and deselect the Green. Once you've verified that only the Red channel is selected, desaturate the selection (RC-->Image-->Colors-->Desaturate).

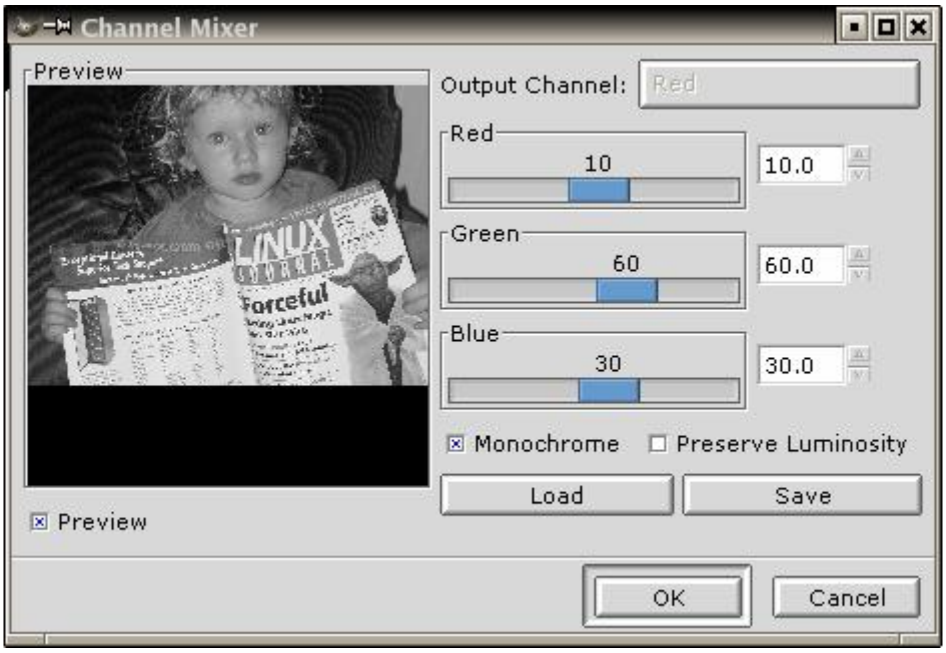
Now it's time to evaluate your results. Press Ctrl-T to toggle the visibility of the selection off; the “marching ants” around the pupils should disappear so you can get a better look. It is important to realize that the selection is still active, just invisible. If you forget to toggle it back on, you can easily forget that you have a selection on the canvas, which can make further edits rather confusing.

When you are satisfied with the results, toggle the selection visibility back on (Ctrl-T), deselect everything (Ctrl-Shift-A) and zoom out (-) to see your handiwork in the unzoomed view and to make further edits. If you are unsatisfied with the result, toggle the selection visibility back on (Ctrl-T) and undo (Ctrl-Z) back to the point where you can make changes in the selection or rectify the problem with some other approach.

I have to mention one variation on this technique that I think gives slightly better results. It requires that you have the Channel Mixer plugin activated in your version of The GIMP.

The Channel Mixer is a great plugin for converting color selections or entire photos to black and white, as it gives you a lot more control over the process than the Desaturate or RGB-->Grayscale conversions. The Channel Mixer wasn't part of my stock Red Hat-based GIMP 1.2.3, but I found it at The Gimp Plugin Registry (registry.gimp.org). Simply compile it and drop it in the .gimp-1.2/plug-ins folder in your home directory.

In this variation you do everything as I mentioned above, but *instead* of the final step of desaturating the red channel, you (re)select *all* of the channels and bring up the Channel Mixer (RC-->Filters-->Colors-->Channel Mixer). The Channel Mixer allows you to mix the RGB values in different percentages. Check the “Monochrome” box and mix down the Red channel significantly and boost Green. I use settings of Red 10%, Green 60% and Blue 30%, as shown in Figure 6. You may need to experiment to see what gives you the most realistic pupils for your photo subjects, but this is a good starting point.



Output Channel: Red
Red 10 10.0
Green 60 60.0
Blue 30 30.0
☒ Monochrome ☐ Preserve Luminosity
☒ Preview

Figure 6. Customized Channel Mixer RGB Values

When you have the mixer settings to your liking, click OK. If you're not sure you like the result and want to try another mix, Undo (Ctrl-Z), toggle the selection visibility back (Ctrl-T) and run the same filter again (Shift-Alt-F). You can see the result of running the Channel Mixer in Figure 7: the pupils look good and dark, with gentle changes in tonality around the edges and a good-looking catchlight in each eye. As before, once you're satisfied with the result, toggle the selection visibility back on (Ctrl-T), deselect everything (Ctrl-Shift-A) and zoom out (-) to see your handiwork in the unzoomed view. Figure 8 shows the final result using the Channel Mixer variation.

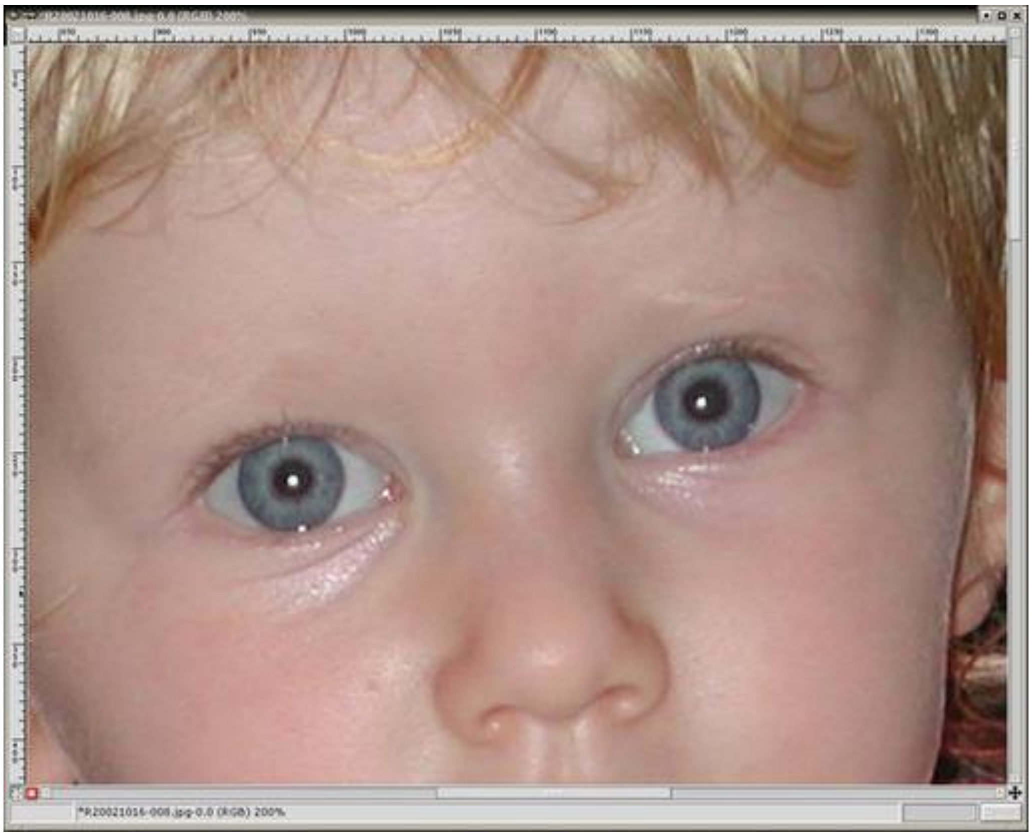
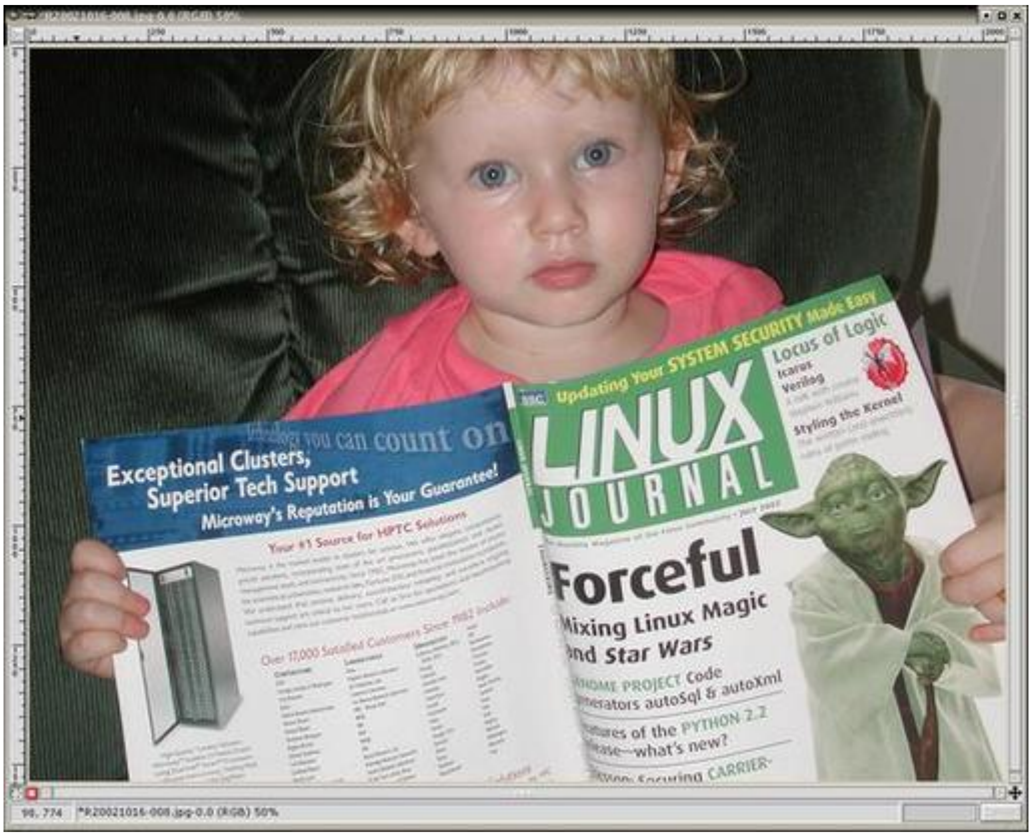


Figure 7. Pupils That Look Human, Thanks to Channel Mixed


Exceptional Clusters, Superior Tech Support
Microway's Reputation is Your Guarantee!
LINUX JOURNAL
Updating Your SYSTEM SECURITY Made Easy
Locus of Logic
Forceful
Mixing Linux Magic and Star Wars
Figure 8. The Final Channel Mixed Photo

This technique may seem slightly involved at first, but it takes only a couple of minutes once you have the hang of it. Best of all, the results are excellent, especially when printed on a high-resolution photo inkjet printer. Throw this technique into your bag of GIMP tricks, and you'll never again have to worry about red-eye ruining your candid flash photos.

email: jeschke@mano.uhh.hawaii.edu

Eric Jeschke (eric@redskiesatnight.com) holds a PhD in Computer Science from Indiana University and has worked as a software engineer, university professor and freelance consultant. He lives in Hawaii with his wife, kids and an overweight cat. Eric enjoys his family, outdoor adventures, taking photographs and running Linux.

Archive Index Issue Table of Contents

Advanced search

A Linux-Based Steam Turbine Test Bench

Alexandr E. Bravo

Issue #106, February 2003

How the Central Boiler and Turbine Institute in St. Petersburg ensures safety and accurate control when testing turbines for power plants.

Despite the fact that mathematical models and the incredible growth in computer power allow one to imitate and calculate almost everything now, there are some areas where real experiments are still very important and can't be replaced with computer models.

One of these areas is the design of low-pressure steam turbines (LPMTs). LPMTs are an important part of any power plant working on a steam or combined gas-steam cycle and generate up to 20% of the power plant's energy. Unlike high- and middle-pressure turbines, where steam has well-known properties, the LPMT works with nonstructured, nonsymmetric wet steam. No fully proved mathematical models exist yet for this kind of flow. Real experiments are crucial for design of the turbine flow path and improvement of the turbine computer models.

There are only a few such test benches in the world. One of them is a part of the Central Boiler and Turbine Institute in St. Petersburg, Russia, where I have worked for the last seven years. Imagine a hall 18 meters in height and 700 square meters in area filled with pipes, wires and measurement equipment. There is a cyclopic construction in the centre (see Figure 1), which is the casing of the model turbine with two huge exhaust pipes. During the tests, it consumes 40 tons of steam per hour, using live steam at a pressure of 30 bars and a temperature of 400°C on the bench inlet, about 4 bars and 200°C on the model turbine inlet and deep vacuum, as low as 30 mbars absolute, on the exhaust.

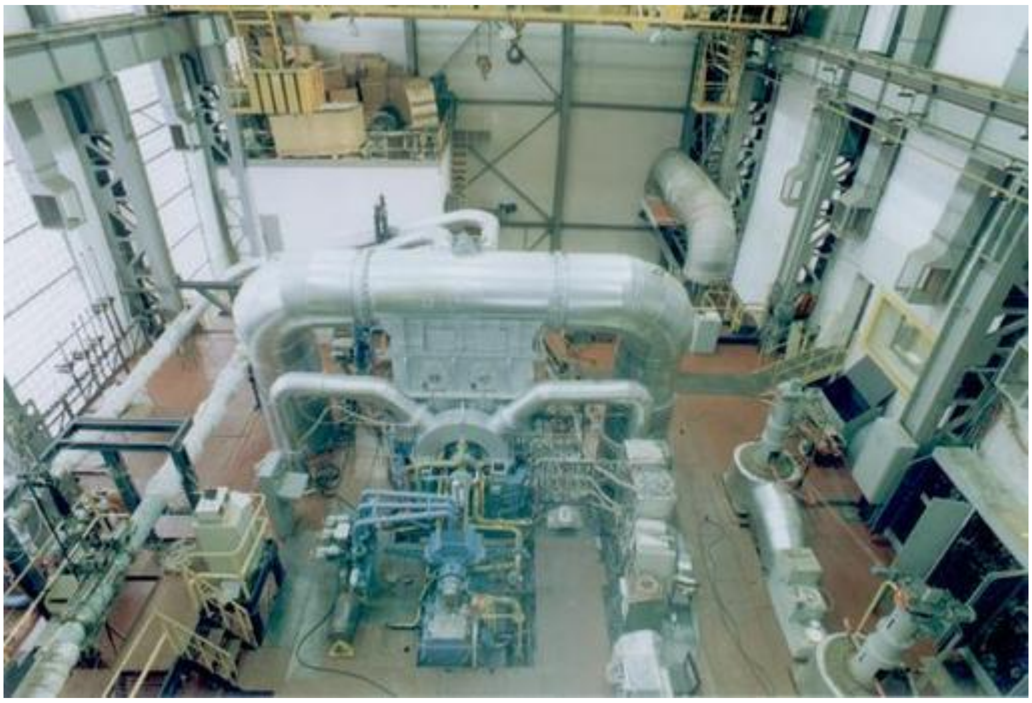


Figure 1. The turbine under test is in the center, surrounded by steam pipes and test equipment.

Computer and Measurement Equipment Structure

During our joint project Tanja with Alstom Power, the information infrastructure of the test bench was renewed. Now it includes three main parts: 1) a high-accuracy scientific measuring system, called Data Acquisition System for flow path measurement, or DAS-Flow; 2) a technological measuring system, called Data Acquisition System for Operational Personnel, or DASOP; and 3) workstations for researchers and engineers.

The DAS-Flow system originally was supplied mostly by our customers. It provides the capabilities to measure more than 200 pressures and 50 temperatures along the flow path. A separate part of this system allows us to investigate the distribution of pressures inside the turbine with 12 movable probes. Each probe can be moved in two directions, axis and angle, by stepper motors through a remote-control system. All the pressure measurements are based on PSI-9000 series pressure transmitters from PSI, Inc. These transmitters provide very high-measurement accuracy: below 0.01% for a few reference pressures and below 0.1% for the rest. The system performs the measurements only from time to time, under stable conditions. It's not designed for dynamic pressure measurements.

The DASOP system was built by ourselves to provide on-line control of the whole bench during the test. It works in real-time mode, collecting data from more than 150 pressure, temperature, speed of rotation and vibration sensors. This data is presented for operational personnel on two monitors and a serial terminal in the control room (see Figure 2) and includes information about the current state of water, oil and steam systems of the bench. DASOP also provides a safety control, with some warning and emergency levels.

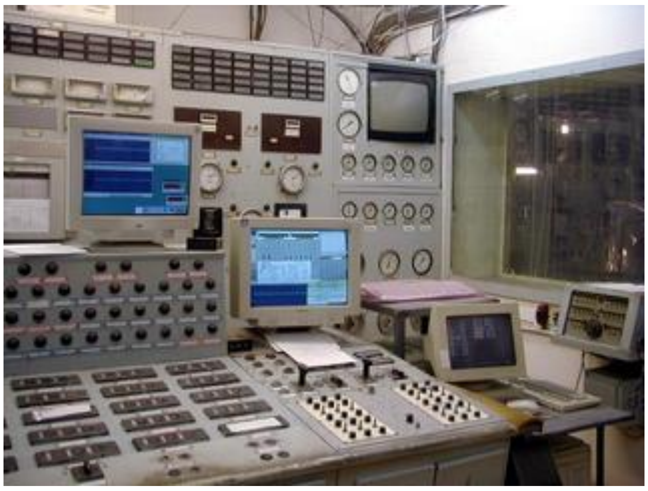


Figure 2. DASOP shows operational and safety information in real time.

All the computers we use, except one IBM RISC workstation, are normal PCs, ranging from 386s up to Pentium 4s and Athlons.

Why Linux?

When I came to the Tanja Project in 1995 I had a lot of practice building data acquisition and evaluation systems based on Russian clones of the Digital PDP-11 running the RT-11 or RSX-11 operation systems. In 1994, I started to play with my first PCs and quickly realized that DOS is not suitable for our tasks because of its single-task and single-user nature. During that time, my brother Mike brought me my first Linux distribution, Slackware as I recall, based on a Linux kernel version somewhere around 0.99. I discovered right away that I could solve almost all of my tasks by studying the sources of similar programs and using them as prototypes. My first data acquisition system was finished in 1994. It was ncurses-based, and it still works for my former employer without any maintenance from me.

At the beginning of the Tanja Project, we had only the DAS-Flow system supplied by our customers. It was a zoo of operating systems. We had MS-DOS, Microsoft Windows 3.11 and NT, QNX and AIX. The positive side was the fact that all the computers were joined on a local TCP/IP network.

Thinking about the development strategy for the whole system and keeping my experience in mind, we decided to stay on Linux as a base for the development of our technological measuring system, DASOP, and for the core of our network. The main reasons were the following:

- The availability of a wide range of ready-to-use applications and the source code of the applications for studying and templating.

- Great stability and reliability on cheap PCs, which is one of the top requirements for our applications.
- We have a very limited budget, so the zero cost of Linux was important.
- A very friendly community available through Fidonet echoes and Usenet newsgroups.

What do we have now? The core of our IT structure is six PCs running Linux. Over more than six years there have been no cases of failure, we have measured uptimes in years, and there have been only two reasons for rebooting: hardware upgrades and long power outages our UPS couldn't handle.

We started with Red Hat, and we still have two computers running Red Hat 4.1 Vanderbilt and Red Hat 5.0 Hurricane. Then we switched to a Russian localized RPM-based distribution named KSI and came to Debian last year. For the moment, our main server and my development machine work under Debian/ Woody. We are very satisfied with Debian, and I think that this year we'll switch all our Linux boxes to Debian.

The Network

All computers are connected to the local network, split into three segments (see Figure 3). The first segment includes all the DAS-Flow computers; the second, the DASOP computers and office computers; and the third looks to the outer world via the leased line. The third segment includes only one computer, which acts as our gateway to the Internet, with a firewall based on ipchains and mail server.

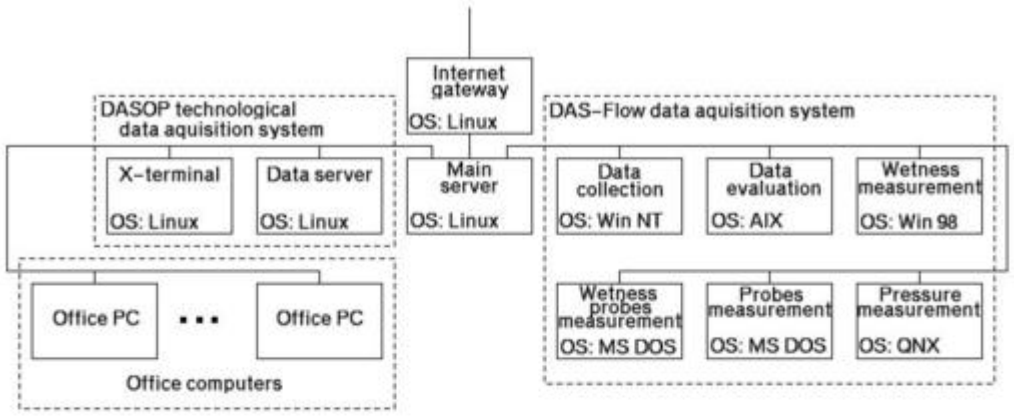


Fig.3 Test bench local network structure

Figure 3. The Test Bench Local Network Structure

In the “middle” of the network is our main server. It acts as file- and print-server for all the computers, but this is not its main task. During the tests, we collect a large amount of data. All this data, as raw, measured values and later as evaluated parameters, is automatically stored in a MySQL database. An Apache web server provides a powerful interface to the database for all users—our local researchers and our customers abroad—through HTTPS.

Any registered user needs to have only a browser to access the database, search the data and get results in a text or graphical form. PNG, CGM and PDF formats are available. We use mostly PHP, as an Apache mod_php module, for generating data-driven pages. Almost all graphs are generated on the fly using the gnuplot program via Perl CGI scripts, which select the parameters from the database, pipe them to gnuplot and then pass the generated image to Apache. We wrote more than 50 different CGI scripts to provide users with all possible kinds of plots, where the user can select everything—what parameters to plot, search conditions, kind of characteristics to plot, auto or manual axes scaling, kind of smoothing and approximation and other choices.

I have to mention specifically the important role of gnuplot in our project. In my point of view, it's one of the greatest scientific plotting utilities with a wide range of capabilities and output formats. It's still under active development, and I'm always trying to use the latest development versions, which are quite stable even for my production environment. I also use the well-designed JpGraph PHP classes for generating certain plots, especially some kinds of fast search results.

Another important part of the software we developed is the technological data acquisition system DASOP (see Figure 4). It has a modular structure and includes the data acquisition module, data evaluation module, socket communication module and application modules.

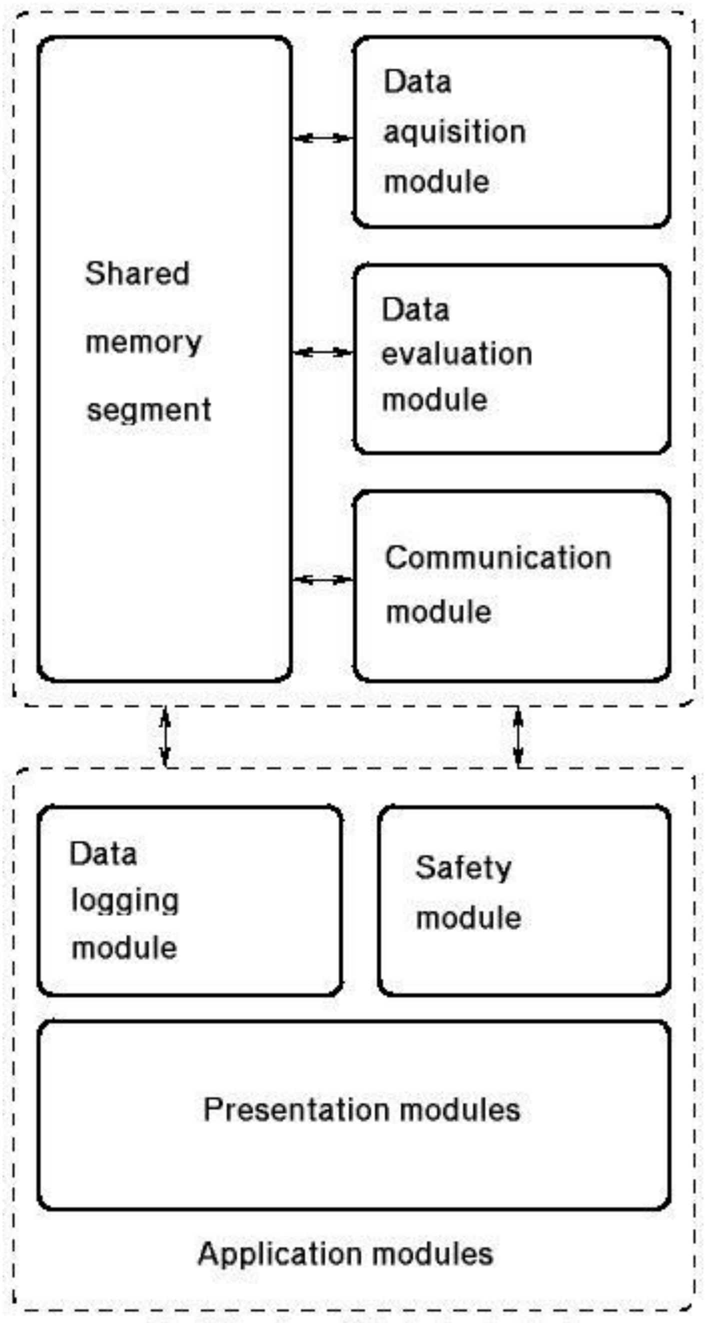


Fig.4 Structure of the technological data aquisition system software

Figure 4. The Structure of the DASOP System

The data acquisition module works with a Programmable Data Controller (PDC) connected by an RS232 interface to a PC. It gets about 150 values from the PDC every second and performs some manipulations with PDC digital I/O if needed. All the measured data is placed in a shared-memory segment as a two-dimensional array, where each column is a full set of all parameters' raw values. The number of columns is fixed, so we always have a fixed number of last-measured datasets in memory.

The evaluation module, which is synchronized with data acquisition modules through a mechanism of semaphores, gets the last measurement set from the shared memory, makes some on-line evaluations and places evaluated data in the same column, extending its length.

The socket communication module provides access to the shared-memory segment for the remote application modules. There are several application modules. Some of them can be run locally with direct access to the shared-memory segment with measured and evaluated data; another can do it remotely via the communication module. Application modules include data logging modules, a safety control module and data presentation modules.

Data presentation modules provide different kinds of graphical presentation of the data in real time. Some examples are parameters over time plots, bar plots (where the color of the bar shows the state of the parameter—normal, warning or emergency) and panels looking like real external devices.

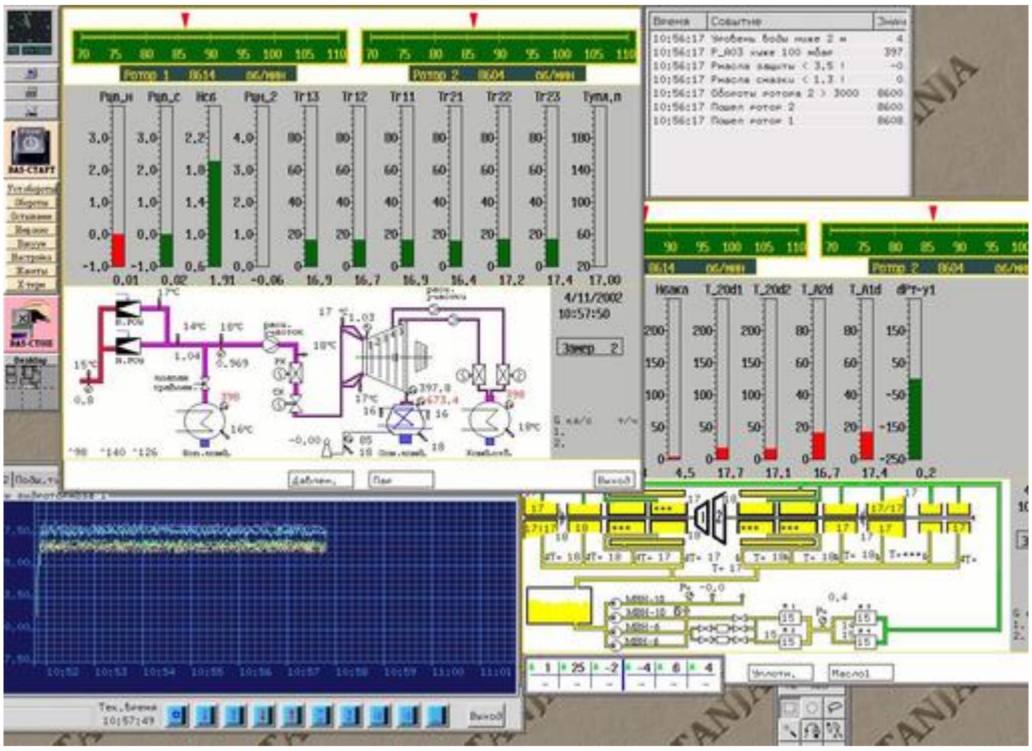


Figure 5. DASOP Data Presentation Modules

Because of our scheduling requirements we do not need hard real-time operation. Soft real time is enough for us, so we use the normal Linux kernel for our hardware. Data acquisition, evaluation and communication modules are written in plain C and work on the same PC. Safety, logging and some of the presentation modules work on that PC also. Part of the presentation modules work on another PC, which works as an X terminal for the first one. Both PCs, with their monitors, are located in the control room of the test bench to provide all the information to operational personnel. Some presentation modules work on the researchers' PCs, getting information via the communication module.

The development of presentation modules has changed over time. At first, they were ncurses-based programs for the Linux text console. Later, I switched to X, using only standard X11 and Xt libraries. The next step was trying Motif, which we bought from SuSE. The open-source GTK appeared one or two years later, and I switched to it. Over the last two years, almost all presentation and other modules have been written in Tcl/Tk, with an extensive use of the BLT extension. I found Tcl/Tk useful for fast GUI development, serial and socket communications and data presentation.

Conclusion

Several years of software development and use in a real industrial environment showed us (and our customers) that open-source solutions are effective from any point of view—cost, time, reliability or function set. Our next steps will be replacing the rest of the proprietary software in our project with open-source software.

Resources

email: abravo@tctube.spb.su

Alexandr E. Bravo was born in 1959 in Leningrad, which is now again (as it was 300 years ago) St. Petersburg, Russia. He graduated in 1982 from Polytechnic University with the specialty “Automation and Telemechanics”. He has worked with Linux since 1994.

The USB Serial Driver Layer

Greg Kroah-Hartman

Issue #106, February 2003

Understanding the USB-to-serial layer and how to get devices into it.

In my last column [see *LJ* December 2002], we covered the serial layer in the 2.5 (hopefully soon to be 2.6) kernel tree. We mentioned in passing that a USB-to-serial driver layer in the kernel helps out in working with those types of device drivers. This time we discuss that layer in more depth.

USB Serial Layer's History

A long time ago (in kernel development time, at least), a single USB-to-serial device driver was written and accepted into the kernel tree. It barely worked for one type of device and didn't work at all on SMP machines. As no standard USB-to-serial protocol existed, all devices used a custom protocol created by the individual vendors. The reason why there is no standard protocol is a long and sordid story; check the archives of the linux-usb-devel mailing list for the details. Soon a second type of USB-to-serial device was implemented within the first driver, sharing the reserved major and minor numbers. Over time, more and more devices were added to the driver until it was becoming an unwieldy mess. With the help of Peter Berger and Al Borchers, the original author of the driver rewrote the infrastructure and created what is now known as the USB-to-serial layer. This bit of code allows different USB-to-serial drivers to be written with a minimal amount of code, all sharing the same major and minor number range. It insulates the individual drivers from some of the complexity in the tty layer and the USB layer. It also allows the drivers to be compiled as individual modules and loaded only when they are needed.

In the 2.5 development cycle, the serial layer was created in order to provide an easier way to write serial port drivers, so as not to have to deal with the tty layer directly. Hopefully, someday the USB and serial layers will be merged. Both maintainers want to see this happen, but they do not have the time to do it. (They would gladly accept patches to accomplish this, if someone is looking for a project.)

In this article we cover the basics of the USB-to-serial layer, detailing how to register and unregister a driver and how to set up the main structures needed for a driver.

Registering and Unregistering a USB Serial Driver

All of the code and examples in this article are for the 2.5/2.6 kernel tree. The 2.4 and 2.2 kernel trees also support USB-to-serial drivers, but their interfaces are a bit different in places. For ease of use, we focus on only one kernel tree. If you have any problems porting a USB-to-serial driver to these older trees (once it is running on 2.5), please let me know.

To register a USB-to-serial driver with the kernel, the driver has to do two things: register with the USB-to-serial core and register with the USB core. Registering with the USB-to-serial core tells it to call your driver when new devices are found by the USB subsystem, and registering with the USB core is needed to tell it what kind of devices your driver can accept.

To register with the USB core, all you need is a list of USB devices that your driver will work for, in traditional USB device ID format:

```
static struct usb_device_id id_table [] = {
        {USB_DEVICE(MY_PRODUCT_ID, MY_DEVICE_ID)},
        {}     /* Terminating entry */
};
MODULE_DEVICE_TABLE (usb, id_table);
```

This table is needed so the USB core knows what devices the driver can accept and the user-space hot-plug code knows what kind of devices are used. See my article “How the PCI Hot Plug Driver Filesystem Works”, *LJ* May 2002, for more information about this table and how the hot-plug code uses it.

Then, a simple USB device-driver structure is created with this ID information:

```
static struct usb_driver tiny_driver = {
        .name =         "tiny",
        .probe =        usb_serial_probe,
        .disconnect =   usb_serial_disconnect,
        .id_table =     id_table,
};
```

The .probe and .disconnect fields must be set to point to the USB serial core's functions because that type of logic is handled by it and not by your driver.

Then, a simple call registers this driver with the USB core:

```
usb_register (&tiny_driver);
```

After this, the USB serial driver must be notified of the driver with a call to:

```
usb_serial_register (&tiny_device);
```

This function takes a pointer to a struct usb_serial_driver_type, which will be explained in the following section.

To unregister a driver, the same steps have to happen, but in reverse order. First, we unregister with the USB serial core:

```
usb_serial_unregister (&tiny_device);
```

Then, we unregister with the USB core:

```
usb_unregister (&tiny_driver);
```

struct usb_serial_device_type Explained

To register with the USB serial core, the usb_serial_device_type structure must be filled. This structure can be found in drivers/usb/serial/usb-serial.h and is defined as the following:

```
struct usb_serial_device_type {
  struct module *owner;
  char *name;
  const struct usb_device_id *id_table;
  char num_interrupt_in;
  char num_bulk_in;
  char num_bulk_out;
  char num_ports;
  struct list_head driver_list;
  int (*probe) (struct usb_serial *serial);
  int (*attach) (struct usb_serial *serial);
  int (*calc_num_ports) (struct usb_serial *serial);
  void (*shutdown) (struct usb_serial *serial);
  int  (*open) (struct usb_serial_port *port,
               struct file * filp);
  void (*close) (struct usb_serial_port *port,
                struct file * filp);
  int  (*write) (struct usb_serial_port *port,
                int from_user,
                const unsigned char *buf,
                int count);
  int  (*write_room) (struct usb_serial_port *port);
  int  (*ioctl) (struct usb_serial_port *port,
                struct file * file,
                unsigned int cmd,
                unsigned long arg);
  void (*set_termios) (struct usb_serial_port *port,
                      struct termios * old);
  void (*break_ctl) (struct usb_serial_port *port,
                    int break_state);
  int  (*chars_in_buffer)
         (struct usb_serial_port *port);
  void (*throttle) (struct usb_serial_port *port);
  void (*unthrottle) (struct usb_serial_port *port);
  void (*read_int_callback)(struct urb *urb);
  void (*read_bulk_callback)(struct urb *urb);
  void (*write_bulk_callback)(struct urb *urb);
};
```

This is a rather large structure, but it's still smaller than either the tty layer's structure or the combination of the serial layer's structures, both of which are alternatives to using the USB serial layer.

Let us describe what all of these fields are used for and whether they are necessary. The owner field is a pointer to the module that owns this device. It should be set to the THIS_MODULE macro. When this is set, the module reference count logic is handled by the USB serial core, which is much safer than trying to do it on your own.

The name field is a pointer to a string that describes this device. This string is used in the syslog messages when a device is inserted or removed. It is also used in the /proc/tty/driver/usb-serial file to show what device is connected to what port.

<u>The /proc/tty/driver/usb-serial File</u>

The id_table field is a pointer to a list of usb_device_id structures that define all of the devices this structure can support. This field can be identical to the pointer that is passed to the USB core. If your driver needs to do different things for different types of devices, however, you can set up different structures describing these devices. An example of this is the Keyspan driver, which handles all of the Keyspan USB serial devices and needs different functions to be called for different devices.

The num_interrupt_in field is the expected number of interrupt in endpoints this device will have. An endpoint is a USB term, defined in the USB spec (<u>www.usb.org</u>). If you do not care about having the USB serial core check for this value (matching it up against any seen devices), use the NUM_DONT_CARE macro defined in usb-serial.h.

The num_bulk_in and num_bulk_out fields state the number of bulk in and bulk out endpoints this device will have. Again, the NUM_DONT_CARE macro can be used here if you do not want the core to pay attention to this value.

The num_ports field indicates the number of different ports this device will have. A single USB serial device can contain many different physical serial ports.

The driver_list field is used by the USB serial core to keep track of all the different drivers registered with it; it should not be used by the individual drivers.

The rest of the fields in the structure are all optional function pointers. If a field is not set, the generic USB serial driver's related function will be called. This allows a driver to be written with a minimal amount of code, if it happens to work the same way as the generic driver does. If not, almost all of these functions will need to be defined. These function pointers are divided into three groups: USB life-cycle pointers, tty life-cycle pointers and urb callback pointers.

The Generic USB Serial Driver

USB life-cycle function pointers consist of probe, calc_num_ports, attach and shut down. They are all called at different points in time as a USB device is initialized and shutdown. The probe function is called when a device matching one of the the id_table devices is inserted into the system. This call happens before the device has been fully initialized by the USB serial core. It can be used to download any needed firmware to the device. In addition, any other early-initialize commands that the device needs can be sent at this time. If 0 is returned, the USB serial core continues on with the initialization sequence. Any other value will abort the call and notify the USB core that this device is not claimed by any drivers.

The calc_num_ports function is called to determine how many ports this device has. This should be used only by devices that can dynamically determine their ports. Any return value overrides the num_ports field in the usb_serial_device_type structure. It is called after the probe function is called but before the attach function is called.

The attach function is called when the struct usb_serial structure is fully set up. Any local initialization of the device or any private memory structure allocation can be done in this function. The shutdown function is called when the device has been removed from the system. Any local memory allocated for this device should be freed up at this time.

tty layer function pointers consist of open, close, write, write_room, ioctl, set_termios, break_ctl, chars_in_buffer, throttle and unthrottle. If you recall the article on the tty layer ["The tty Layer", *LJ* August 2002],the these match up with the tty layer function call of the same name, with a few twists. First off, they all pass in a pointer to the specific usb_serial_port structure that is being operated on, and some of the functions are only called when something needs to happen.

The open function is called the first time open() is called on a port, but not for any subsequent calls to open(). Any urb submission the driver needs to do to start receiving data, or any device-specific messages that should be sent, should be done at this time. If any errors occur, they should be returned; otherwise, return 0 to signal success.

The close function is called for the last close() call, which is called from user space. Any running urbs should be shut down, and any device-specific commands that are needed should be sent now.

The write function is called exactly like the tty layer write function is called. The data passed to the function needs to be sent to the specified port. The number of bytes sent to the device should be returned. Remember, the device does not have to send all of the data that the user wants it to; a short write can happen, as long as the driver notifies user space that this has happened. This allows the driver logic to be much simpler. If an error happens, it should be returned as a negative number.

The write_room and chars_in_buffer functions are closely related. The write_room function is called by the tty layer to ask how many bytes the driver can accept to be written out at this time. The chars_in_buffer function is called to find out the number of bytes still left to be sent to the device.

The ioctl function is called with a wide range of tty ioctl values. If the driver cannot handle the specific value, -ENOIOCTLCMD should be returned. This will allow the tty layer to try to provide a default function. Some of the more common values asked for by user space are documented in the tty driver article previously mentioned.

The set_termios function is called to set terminal settings for a specific port. This includes baud rate, flow control, data bits and other line settings. The break_ctl function is called to set the BREAK value for the port. A state of -1 means that the BREAK status should be turned on, and a status of 0 means it should be turned off. The throttle and unthrottle functions are used to stop and resume data being received from the serial port.

urb Callback Function Pointers

The read_int_callback, read_bulk_callback and write_bulk_callback function pointers are all used by the USB serial core to set up the initial callbacks for these kinds of USB endpoints. If the driver does not specify the read or write bulk callback functions, the generic callbacks are used. There is no generic read interrupt callback function, so if your device has an interrupt endpoint, you must provide this callback.

The operation of the generic read bulk callback adds the data received by the USB urb to the port's tty buffer, to be sent to user space when read() is called. It then resubmits the urb to the device. If your device does not need to interpret the data received in any way, I recommend using this function instead of writing a new one. The generic bulk write callback is much smaller and only wakes up the tty layer (in case it was sleeping, waiting for data to be transmitted to the device).

Conclusion

In this article we have explained how to register and unregister a USB serial driver, as well as the basics of the main usb_serial_driver_type structure that all USB serial drivers must provide. Next time, we will go into the specifics of how the USB serial driver layer works and provide an example driver.

Acknowledgements

I would like to thank all of the different programmers who have helped to create the USB serial layer over the years. Special thanks to Peter Berger and Al Borchers for their loadable module code offered back in July 2000, which is still in place today.

Greg Kroah-Hartman is currently the Linux USB and PCI Hot Plug kernel maintainer. He works for IBM, doing various Linux kernel-related things and can be reached at greg@kroah.com.

The Linux USB Input Subsystem, Part I

Brad Hards

Issue #106, February 2003

As the USB input subsystem spreads further with each kernel release, it's time to understand what it's doing for your devices.

The Linux USB input subsystem is a single, harmonized way to manage all input devices. This is a relatively new approach for Linux, with the system being partly incorporated in kernel version 2.4 and fully integrated in the 2.5 development series.

This article covers four basic areas: a description of what the input subsystem does, a short historical perspective on development, a description of how the input subsystem is implemented in the kernel and an overview of the user-space API for the input subsystem and how you can use it in your programs. The first three areas are discussed in this article. The user-space API, the final topic, will be discussed in Part II of this article.

What Is the Input Subsystem?

The input subsystem is the part of the Linux kernel that manages the various input devices (such as keyboards, mice, joysticks, tablets and a wide range of other devices) that a user uses to interact with the kernel, command line and graphical user interface. This subsystem is included in the kernel because these devices usually are accessed through special hardware interfaces (such as serial ports, PS/2 ports, Apple Desktop Bus and the Universal Serial Bus), which are protected and managed by the kernel. The kernel then exposes the user input in a consistent, device-independent way to user space through a range of defined APIs.

How We Got Here

The Linux input subsystem is primarily the work of Vojtech Pavlik, who saw the need for a flexible input system from his early work on joystick support for

Linux and his later work on supporting USB. The first integration for the input subsystem replaced existing joystick and USB drivers in the 2.3 development kernel series. This support carried over to version 2.4, and input support in the 2.4 series is basically limited to joysticks and USB input devices.

The 2.5 development kernel series fully integrates the input subsystem. This tutorial is based on the full integration, which will be the input API for the 2.6 stable kernel. Although some differences exist in the user-space APIs between 2.4 and 2.5 kernels at the time of this writing, there is ongoing work to harmonize them—mainly by updating the 2.4 kernel.

Under the Hood—Understanding the Kernel Internals

The three elements of the input subsystem are the *input core*, *drivers* and *event handlers*. The relationship between them is shown in Figure 1. Note that while the normal path is from low-level hardware to drivers, drivers to input core, input core to handler and handler to user space, there usually is a return path as well. This return path allows for such things as setting the LEDs on a keyboard and providing motion commands to force feedback joysticks. Both directions use the same event definition, with different **type** identifiers.

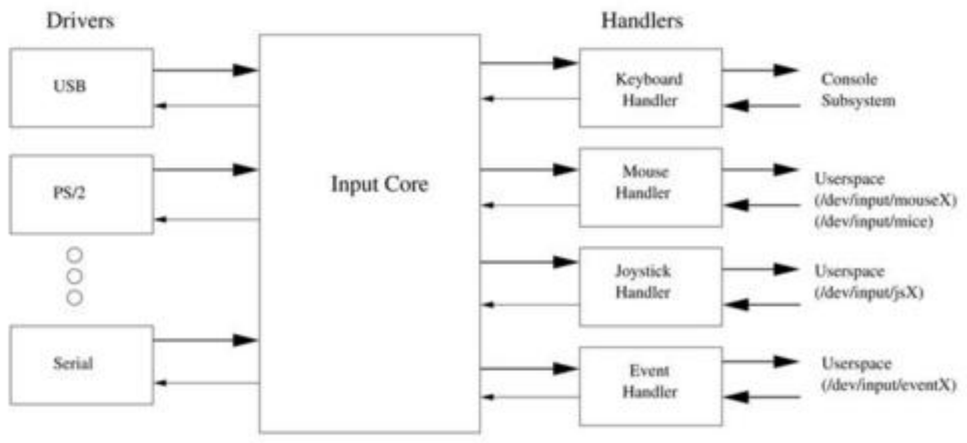


Figure 1. The Relationship between the Elements of the Input Subsystem

The interaction between various elements is through *events*, which are implemented as structures (see Listing 1). The first field (time) is a simple timestamp, while the other fields are more interesting. The type field shows the generic type of event being reported, for example, a key press or button press, relative motion (like moving a mouse) or absolute motion (like moving a joystick). The code field tells which of the various buttons or axes are being manipulated, while the value field tells you what the state or motion is. For example, if the type is a key or button, code tells you which key or button it is, and value tells you if the button has been pressed or released. Similarly, if type is a relative axis, then code tells you which axis, and value tells you how much motion you have and the direction of that motion on that axis. If you move a mouse in a diagonal direction, while simultaneously moving the scroll wheel, you will get three relative events per update: one for motion in the vertical direction (Y-axis), one for motion in the horizontal direction (X-axis) and one for motion of the wheel.

Listing 1. event-dev-struct.txt

Event handlers provide the interface to user space, converting the standard event format into the format required by a particular API. Handlers usually take care of the device nodes (/dev entries) too. The most common handler is the keyboard handler, which is the “standard input” that most programmers (especially C programmers) are familiar with.

Drivers usually interface with low-level hardware, such as USB, PCI memory or I/O regions, or serial port I/O regions. They convert the low-level hardware version of the user input into the standard event format before sending it to the input core. The input core uses a standard kernel plugin design, with input_register_device() used to add each device and input_unregister_device() used to remove it. The argument to these calls is the input_dev structure, which is shown in Listing 1. Although this structure looks quite large, most of the entries are provided to allow a driver to specify the capabilities of the device, such as which event types and codes may be sent or received by the device.

In addition to managing drivers and handlers, the input core also exports a useful /proc filesystem interface, which can be used to see what devices and handlers are currently active. Here is a typical example from /proc/bus/input/devices showing a USB mouse:

```
I: Bus=0003 Vendor=046d Product=c002 Version=0120
N: Name="Logitech USB-PS/2 Mouse M-BA47"
P: Phys=usb-00:01.2-2.2/input0
H: Handlers=mouse0 event2
B: EV=7
B: KEY=f0000 0 0 0 0 0 0 0 0
B: REL=103
```

The I: line is the identity information—showing bus type 3 (which is USB) and the vendor, product and version information from the USB descriptors in the mouse. The N: line shows the name, which in this case is a string provided by the USB descriptors. The P: line shows the physical device information; here, it's structure information comprised of the PCI address for the USB controller, the USB tree and the input interface. The input0 part indicates this is the first logical input device for the physical device. Some devices, such as multimedia keyboards, can map part of the physical device to one logical input device and map another part to a second logical input device. The H: line shows the handler drivers associated with this device; we'll discuss this in more detail later in the article. The various B: lines show the bitfields that identify the devices' capabilities, in this case some keys for the buttons and relative axes for the ball and the scroll wheel.

Listing 2. register.c

This /proc interface is a useful way to test some simple drivers. Let's consider the example of a driver that registers on init and unregisters on removal, as shown in Listing 2. This does some preliminary initialization using init_input_dev(). It sets up the name, physical and identification descriptors, and then sets up the bit arrays to indicate that the device is capable of providing one type of event (EV_KEY indicating buttons and keys) with two possible codes (KEY_A and KEY_B, indicating the key labels). The initialization routine then registers the device with the input core. If you add this code to the kernel (using **modprobe**), you can see the new device has been added to /proc/bus/input/ devices, as shown below:

```
I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="Example 1 device"
P: Phys=A/Fake/Path
H: Handlers=kbd event3
B: EV=3
B: KEY=10000 40000000
```

If we actually want to send events from our device driver to the input core, we need to call input_event or one of the convenience wrappers, such as input_report_key or input_report_abs, provided in <linux/input.h>. An example of code that does this is shown in Listing 3. This example is basically the same setup as the previous one, except that we add a timer that calls ex2_timeout(). This new routine sends four presses of KEY_A and four presses of KEY_B. Note that 16 key press events are created in total, because a separate event is created for each press and each release. These events are passed to the input core and, in turn, to the keyboard handler, which will cause the pattern “aaaabbbb” or “AAAABBBB”, depending on the Shift key selection, to be transmitted to the console or command line. The timer is then set up to run four seconds later, looping infinitely. The four-second delay is intended to give you enough time to remove the module when you have seen enough of the pattern. If you reduce the delay, make sure you have another way of accessing the system, such as an SSH connection. Also note the call to the input_sync function. This function is used to inform the event handler (in this case, the keyboard handler) that the device has transmitted an internally consistent set of data. The handler may choose to buffer events until input_sync is called.

Listing 3. aaaabbbb.c

Let's look at one final example of a driver, this time showing how relative information is provided, shown in Listing 4. This example is a driver that emulates a mouse. The initial setup configures the device to have two relative axes (REL_X and REL_Y) and one key (BTN_LEFT). As in the previous example, we use a timer to run **ex3_timeout**. This timer then calls input_report_rel to provide small relative motion (five unit steps—the relative movement is the third argument to the function) consisting of 30 steps right, 30 steps down, 30 steps left and 30 steps up, so the cursor is moved in a square pattern. To provide the illusion of movement, the timeout is only 20 milliseconds. Again, note the call to input_sync, which is used to ensure that input handlers only process events that make up a consistent set. This specification wasn't strictly necessary in our previous example. But, it is definitely required to convey information like relative movement to the input core, because more than one axis may be required to represent movement. If you were moving diagonally, you would do something like:

```
...
input_report_rel(..., REL_X, ...);
input_report_rel(..., REL_Y, ...);
input_sync(...);
...
```

which ensures that the mouse will move diagonally and not across, then up.

Listing 4. squares.c

Handlers—Getting to User Space

In the previous section, we saw that the device drivers basically sat between the hardware and the input core, translating hardware events, usually interrupts, into input events. To make use of those input events, we use handlers, which provide a user-space interface.

The input subsystem includes most of the handlers you'll likely need: a keyboard handler to provide a console, a mouse handler for applications like the X Window System, a joystick handler for games and also a touchscreen handler. There is also a general-purpose handler called the event handler, which basically provides input events to user space. This means you almost never need to write a handler in the kernel, because you can do the same thing with the event handler and equivalent code in user space. This API discussion is covered in the second part of this article.

Acknowledgements

I'd like to thank Greg Kroah-Hartman and Vojtech Pavlik for their assistance with this article.

Resources

email: bhards@bigpond.net.au

Brad Hards is the technical director for Sigma Bravo, a small professional services company in Canberra. In addition to Linux, his technical focus includes aircraft system integration and certification, GPS and electronic warfare. Comments on this article can be sent to bradh@frogmouth.net.

Choosing Tools

Reuven M. Lerner

Issue #106, February 2003

The pros and cons of four web development tools: mod_perl/Mason, J2EE, Zope and OpenACS.

If someone asked you to name the best car on the market, you'd probably tell them the answer depends on who will use the car. After all, a family of eight living in Manhattan probably needs a different type of vehicle from a single hacker living in rural North Dakota. The same is true for programming languages and development toolkits. Each has its place and is appropriate for solving different sorts of problems.

Although this might seem obvious, many programmers believe the language or toolkit they use is so good it should be used to solve all problems, all of the time. As the old saying goes, if your only tool is a hammer, every problem looks like a nail. No programming language is the best fit for all problems, which is why experienced programmers know and use a variety of languages and constantly are learning new ones.

Until only a few years ago, programmers were largely concerned with optimizing their programs for speed and memory usage. After all, processors were relatively slow, and RAM was fairly expensive, so any program that didn't try to squeeze the most out of the hardware was seen as bloatware.

Today, however, we are blessed with cheap, fast computers and cheap, plentiful RAM. This means that software engineers can and should use languages that encourage rapid development and long-term program maintenance. I'm not against optimizing programs for speed or memory usage, but they are less important than creating stable, maintainable software quickly and easily.

About two years ago, I decided I would devote a long series of columns to four basic web development technologies: mod_perl/Mason, J2EE, Zope and

OpenACS. These technologies are not only interesting and useful but thought-provoking as well, providing new perspectives and ideas for web developers. And although occasional arguments arise among these communities about which product is superior, the fact is that each tries to solve a slightly different type of problem.

This month, we take some time to summarize the ideas and frameworks that we've explored over the last few years. I don't expect that everyone reading my column will jump to use all of the technologies I name here; however, I do hope to provide even the most die-hard aficionados with some food for thought.

mod_perl/Mason

Apache is deservedly one of the poster children of the Open Source movement. It is reliable, highly configurable, well documented, stable and extensible. You can do amazing things with Apache and can customize it in any number of ways to fit your needs. If you are writing a web application that must execute as quickly as possible, you can write a new module in C that seamlessly hooks into Apache.

Although C programs execute quickly and Apache libraries (now known as the Apache Portable Runtime) provide a great deal of useful infrastructure and support for module authors, development in C is slower and more prone to bugs than working with a high-level language such as Perl or Python. So it shouldn't come as any surprise that there are Apache modules that embed these languages inside of the Apache server. mod_perl allows you to write Apache modules in Perl, rather than C, giving you nearly unlimited control over your server with all of the speed and flexibility of Perl.

And indeed, mod_perl comes to mind whenever someone asks me to create a high-performance web application, particularly one that involves text processing or a relational database. I could write the module in C, but why bother? There are times when it makes sense to code in C, but I've generally found mod_perl to be fast enough for even high-powered applications.

Of course, the wonder of mod_perl diminishes somewhat when graphic designers enter the scene. Designers have no interest in modifying program code whenever they want to change the style (or content) on a given page, and giving them access to the source code of Perl modules is asking for trouble. Thus, dozens of different templating systems are available, each of which allows you to mix Perl and HTML in a slightly different way. One of the most popular is Mason, which has been used on a large number of heavy-duty publishing sites for years.

Mason is indeed a wonderful tool, and it provides an excellent trade-off between rapid development (thanks to Perl), easy maintenance (thanks to Mason) and fast execution (thanks to mod_perl). The Mason e-mail list is a source of useful information and support, and the package maintainers have done an admirable job of improving it steadily over time. Configuring, using and debugging modern versions of Mason make the versions that I first used several years ago appear primitive in comparison.

At the same time, Mason is an infrastructure and framework for creating your own applications. True, you easily can use Apache::Session to generate cookies and associate users with a unique ID, but anything having to do with user registration, groups and permissions, let alone full-fledged applications, are your responsibility to implement. For some projects, this is just fine, because it gives you the flexibility you may need. But the fifth time you find yourself creating a system for creating and managing users, groups and permissions, you may decide you need something with a bit more infrastructure.

Java and J2EE

Sun has been pushing Java as a server-side solution for several years now, and J2EE (Java 2, Enterprise Edition) is the umbrella for a variety of technologies that are meant to help developers create such solutions. Servlets are classes that execute code on a server; JavaServer Pages (JSPs) are Java/HTML templates that are compiled into servlets on the fly. JDBC allows you to access the database, and Enterprise JavaBeans provide you with transactions and automatic relational-to-object mapping. Entering the world of Java requires learning a huge number of acronyms and technologies, as well as learning the various versions for different standards.

I've been working with Java at various times since it was first released, and on nearly every occasion, I find myself wanting to get excited about it but being unable to do so. Java isn't bad, per se, and the different technologies it brings to the table are all rather impressive. Servlets are easy to write; JSPs (and especially the custom tags you can create for JSPs) are a mature and impressive templating system, and JDBC provides everything you would ever want in a database interface. And although EJB is undoubtedly overkill for most projects, it is extremely useful for the big enterprise development groups that Sun is targeting. In addition, multiple implementations, including fine open-source application servers and tools, are impressive and encouraging.

Indeed, Java seems to be the “big company” of the web development world. It gets things done reliably; it has an enormous array of talent at its disposal and follows a huge number of standards; oodles of development tools are available, and a lot of people are using Java. But the overhead associated with Java projects is too large for my liking. Simply learning which version of which standard goes with which version of which Jakarta subproject can take a fairly long time. Just as it's typically more fun to work at a small company than a large one, I find it more interesting to program in Perl or Python than in Java.

Moreover, J2EE suffers from problems similar to those I described with mod_perl and Mason, namely the fact that it's purely infrastructure, without any attention paid to built-in applications. Developers can create amazing things but must reinvent the wheel for each project.

Perhaps my favorite part of the Java world is the attention to maintainable and reliable software. A fair number of testing and development tools, such as Ant, Cactus, JUnit and log4j make it possible (and even relatively straightforward) for programmers to create and manage comprehensive testing of software before it is released.

So, is Java a good choice for web development? I would argue that the larger your project, the more seriously you should consider Java. But for the typical basic web application that small shops work on, the overhead associated with development is too great to ignore.

Zope

Zope is clear proof that open-source software does more than imitate its proprietary competition. Zope combines an object database with a multiprotocol server, hooking them together with a rich set of objects and a slick web-based management interface. Zope is innovative, clever, a pleasure to work with and one of the rare open-source projects designed with end users, not just hackers, in mind. Graphic designers love to hear that they can revert to any previous version of a document by using the undo feature in the web-based management interface.

Zope has a number of programming interfaces, each of which trades off simplicity for power. You can create simple DTML templates and Python scripts, use the fascinating ZPT templates that completely separate programs from the display logic, or you can go all the way and create a new Zope product. Zope products are where the real power is, and because each product is a class, you can create multiple instances of your product at different URLs. Because objects inherit via the URL hierarchy (acquisition) in addition to their native object hierarchy, the permissions, behavior or look and feel of a product instance can vary according to its URL.

So far, it sounds like Zope is the best thing that happened to the Web since HTTP. And indeed, the growing number of Zope hackers means a large number of products are available for free download, as well as a growing number of commercial products that use Zope as their underlying infrastructure.

However, Zope has a few problems, the first and biggest one being that the learning curve can be rather steep. Even if you're an experienced web developer, Zope requires that you re-learn nearly all of the concepts from scratch, changing almost all of the habits you've acquired over the years. This isn't necessarily a bad thing, as Zope implements it so well, but it can be a surprise and a reason to be wary, simply because using Zope inevitably will slow things down during the initial startup period.

The other issue I have with Zope is its object database. Object databases historically have had a lot of problems, and ZODB appears to be bucking that trend nicely. At the same time, relational databases are still pretty standard, and people expect (and often need) to work with them. In theory, this isn't a problem. Zope's built-in ZSQL methods allow you to do fascinating things with relational database queries without thinking very hard. The problem then is that your data is split across two different locations: ZODB and your relational database. I like to keep all of my data in one central location, which means this split can frustrate me somewhat.

There is also the issue of speed. Zope's sophisticated permissions and acquisition mechanism is probably faster than you or I could implement on our own, but it still can be relatively sluggish. The official Zope solution for this problem is ZEO (Zope Enterprise Objects), which allows multiple Zope servers to access a single object database. This apparently scales to one million hits per day, which is more than adequate for most of the sites I work on. But exceptionally large sites may need to worry about how quickly Zope operates or alternatively, consider investing in some high-end hardware for the central ZODB server.

Finally, Zope products tend to be relatively independent. The good news is that this allows developers to work in parallel, without slowing each other down. The bad news is that things are not as unified as they could be, with repeated functionality and a lack of coordination. This may be inevitable in an open-source project of this magnitude, but it can be frustrating at times.

Over the last year or two, Zope Corporation has been pushing the use of Zope for content management, rather than for application development. Of course, any decent content management system needs to be modified and reprogrammed for the needs of the customer, so the difference isn't that pronounced. Although Zope is not the only open-source content management system on the market, it is undoubtedly one of the most sophisticated, as well as one of the most mature.

In my own work, I pitch Zope to clients whose projects will involve a fair amount of tricky development, on those that require a relatively easy to use interface or those that require content management. I continue to be impressed by it and look forward to working with Zope quite a bit in the years to come.

OpenACS

OpenACS began as a sophisticated data model for community web sites, along with a large number of web/database templates written in Tcl. Over time, it has grown into a much larger project with a number of facets: independent packages that can upgrade both programs and the data model, the ability to work seamlessly with either Oracle or PostgreSQL and a sophisticated templating system that separates programs from the HTML output. And, OpenACS comes with a huge number of prebuilt applications that do about everything you would want for a community web site, from weblogs to fora and FAQs to an ecommerce store. With nothing more than your web browser, you can create a site in very little time.

And indeed, I find myself recommending OpenACS again and again to nonprofits that want to create on-line communities, reach out to their constituents, conduct discussions among the members or publicize information easily, without needing to know much in the way of technology.

That said, OpenACS has a number of issues. First and foremost is the learning curve. Zope's learning curve is difficult because there are so many technologies to understand. OpenACS has a much simpler model, but it stores absolutely everything in a relational database. This means the data is all in one place, but relational databases are notoriously bad at dealing with hierarchies, and all of the clever OpenACS developers in the world cannot mask that.

Thus, learning to work with OpenACS requires that you learn how to implement a simple object inheritance system and the extensive API that allows you to do it. If you haven't ever written stored procedures or worked with databases containing dozens or hundreds of tables, then you may be overwhelmed by the knowledge necessary to work with OpenACS.

OpenACS also suffers from little documentation for developers and none for end users. OpenACS is admittedly a complex system that can be difficult to describe to nontechnical people, but it can be maddening to find nothing to help with that. To their credit, the main openacs.org site was recently remodeled and rewritten shortly before I wrote this article and seems to have made some positive headway in this direction.

Finally, I find it somewhat ironic that OpenACS has become increasingly sluggish over time. Granted, this is because the latest version (4.x, as of this writing) is far more clever about users, groups and permissions than its predecessors, and checking these things with each HTTP request takes time. In addition, the developers know many optimizations still can be made, such that each request doesn't require quite so many database queries.

Conclusion

Several days before I wrote this article, a new report appeared on the Web describing how Yahoo has settled on PHP as a web programming environment. I personally prefer to work in other languages and wouldn't relish the idea of rewriting all of Yahoo in a new language. But for Yahoo's particular needs, it seems like PHP is indeed a good choice. I give them credit for considering all of the options, weighing the pluses and minuses and coming to a conclusion that fit their needs.

As I said at the beginning of this article, I am a firm believer in finding a technology that meets the needs of the problem at hand. As a developer, this means I'm constantly forced to learn new languages, technologies and techniques. At the same time, this means my clients can get a solution that's appropriate for their problems, and I gain broader skills and depth as a software engineer.

The fact that these systems are available free of charge via the Internet means that the only thing stopping you from trying them is time and some effort. I strongly encourage you to find the time to work with them; you and the people you work with will undoubtedly enjoy the results.

Resources

email: reuven@lerner.co.il

Reuven M. Lerner is a consultant specializing in web/database applications and open-source software. His book, Core Perl, was published in January 2002 by Prentice Hall. Reuven lives in Modi'in, Israel, with his wife and daughter.

Charting the Enterprise

Marcel Gagné

Issue #106, February 2003

These two open-source project management tools can help you track your progress on any undertaking.

There is it, François, on page 509 of the *Collins* dictionary. *Enterprise* (the focus of this month's issue) is a word that comes from the French word *entreprise*. You see, *mon ami*, it is as I told you—a project or undertaking that requires boldness or effort. These days, of course, we often refer to an enterprise as a large-scale business organization. Nevertheless, the definition still stands.

People take on enterprises (or projects, if you will) every day. The success of these enterprises often is tied closely with how the project is managed, how carefully it is planned and how diligently the progress is tracked.

Mon Dieu, François! Why did you not tell me it was so late? Our guests are already here. *Bonjour mes amis*, and welcome to *Chez Marcel*, home of fine Linux fare and devastatingly good wine. François! Bring up the 1996 South Australia Coonawarra Shiraz *immédiatement*!

As I was telling François, *mes amis*, the focus of today's menu deals with enterprise, the taking on of bold, new projects. Aside from the obvious project management aspects of beginning any enterprise, there seems to be a great deal of excitement when it comes to charting the status of any large project. This usually is done with Gantt charts, which were developed in 1917 by Henry L. Gantt. The familiar horizontal bar chart was developed as a production control tool to provide a quick visual means of determining where and how a project was going, thereby simplifying project management and tracking.

Here's how it works. A Gantt chart's horizontal axis represents the time for the project. This can be broken up into days, weeks or whatever time period makes sense. After all, *mes amis*, some projects can last an awfully long time. The vertical axis lists tasks that make up the project. For instance, I have been restocking the wine cellar here at *Chez Marcel*. My task list involves an inventory (François), some amount of shelf reorganization (François), quality control and tasting (Chef Marcel), placing the orders for more wine (François and Chef Marcel), delivery (Henri of Henri's Fine Wines) and finally, restocking (François).

Speaking of François, our esteemed waiter has returned. François, please pour for our guests. You will love this Coonawarra Shiraz, *mes amis*. In addition to the traditional spiciness of the Shiraz, notice the chocolate in its bouquet, perhaps even a little mint, *non*? And the taste...but I digress. I was talking about Gantt charts.

Tasks are listed with horizontal bars of varying lengths (and color perhaps) to represent the amount of time spent on the specific task. At any point in the cycle, you can draw a vertical line from the top of the chart (more or less) to create a report of where the project stands. Simple, *non*?

Simple is the idea behind the Graphical UI Gantt Chart Generator (originally by Jason R. Govig and Seth Goldstein, now maintained by Glen Stewart). This web-based system for generating Gantt charts is simply a collection of CGI scripts. It is perfect if you are looking for something that allows for easy, network-available charting.

It requires Apache and a couple of Perl modules: CGI.pm and Date::Manip.pm. The easiest way to install these Perl modules is at the command line using CPAN:

```
perl -MCPAN -e "install Date::Manip"
perl -MCPAN -e "install CGI"
```

I should tell you that you need to do this as root. If you have never run a Perl CPAN install before, you will be asked a number of setup questions. This only happens once, *mes amis*--other than that, it is a very smooth process. To use the package, point your web browser to the appropriate URL. On my system, it looked something like this: **http://webserver/gantt/**.

The source for the Gantt chart package is available at associate.com/gantt. To start, extract the source into your web server's document root:

```
tar -xzvf gantt-1.0.tar.gz
mv gantt-1.0 gantt
```

Notice that I immediately renamed the directory to something easier. This can be anything you like. Have a look at the directories under the distribution directory, specifically the one called users. This entire directory must be writable by whatever user you have Apache running under (on my system, the user is www). Look for the user and group in your httpd.conf file.

Some modifications are necessary for two files in particular. The first is variables.pl. The lines to change represent your own site configuration, including the document root (as discussed above), the URL to the chart generator, the admin's name and the admin's e-mail address:

```
# full path to site on server
$docroot = '/usr/local/apache/htdocs/gantt/';
# URL of site
$wwwroot = 'webserver.yourdomain.dom/gantt/';
# Name of site administrator
$admin = '
# Email of site administrator
$adminEmail = 'your_email@yourdomain.dom';
```

You'll also need to make one small modification to the dbhelp.pl file, providing it with the path to variables.pl:

```
# Edit this to point to the location of your
# variables.pl file
require \
'/usr/local/apache/htdocs/gantt/variables.pl';
```

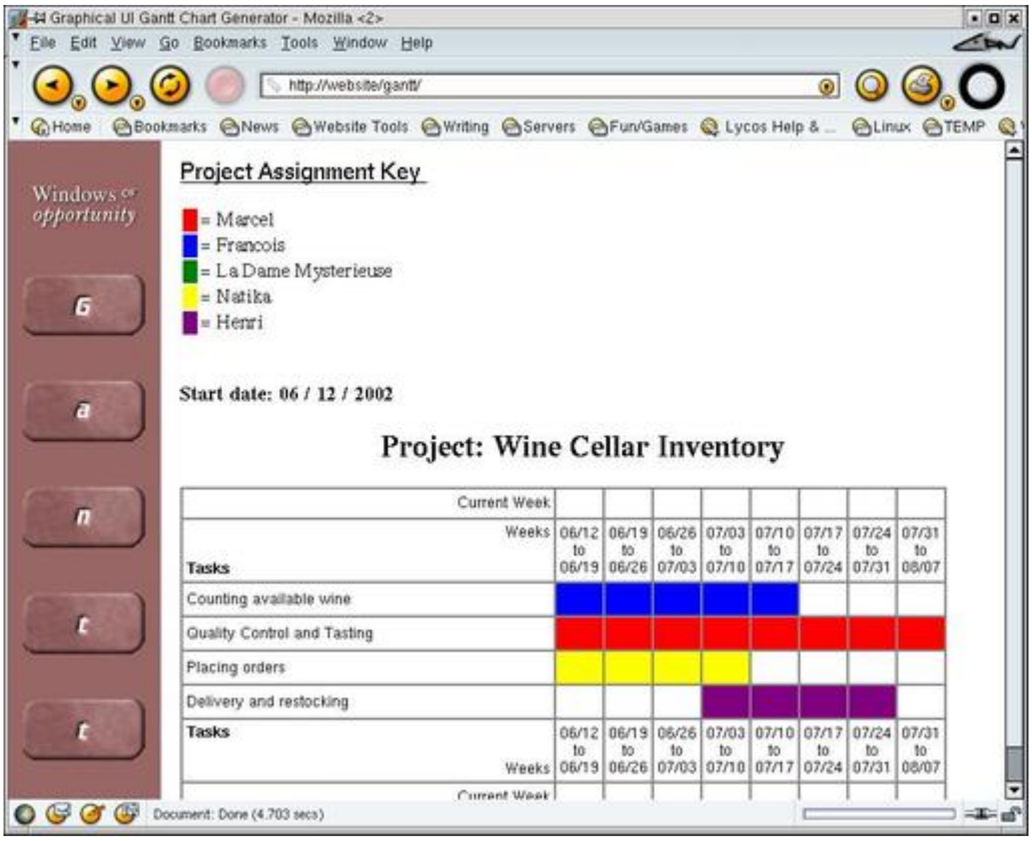
Finally, your Apache server's httpd.conf needs a small edit. To allow CGI scripts to execute from the gantt directory (which lives under document root), you need something akin to the following paragraph:

```
<Directory "/usr/local/apache/htdocs/gantt">
    Options ExecCGI
</Directory>
```

When you restart your web server (**apachectl graceful**), you'll be ready to roll.

To use the Gantt chart generator, enter a name into the login field—the form actually asks for an e-mail address, but any unique name will do. If this is the first time through, you'll be presented with a dialog form to enter your name and contact information. After you click the Submit button, you can describe your project and identify the members of your project team.

Next, you'll list the tasks that will bring about the successful completion of your new enterprise. Each line is color-coded and new tasks can be added at any time. With each update of the page, enter the starting and ending week and the individual responsible. When you finally click Submit, the chart is generated automatically (Figure 1).


Graphical UI Gantt Chart Generator - Mozilla <2>
File Edit View Go Bookmarks Tools Window Help
http://website/gantt/
Home Bookmarks News Website Tools Writing Servers Fun/Games Lycos Help & ... Linux TEMP
Windows of opportunity
G
a
n
t
t
Document: Done (4.703 secs)

Project Assignment Key

🟥 = Marcel
🟦 = Francois
🟩 = La Dame Mysterieuse
🟨 = Natika
🟪 = Henri

Start date: 06 / 12 / 2002

Project: Wine Cellar Inventory

Current Week								
Weeks Tasks	06/12 to 06/19	06/19 to 06/26	06/26 to 07/03	07/03 to 07/10	07/10 to 07/17	07/17 to 07/24	07/24 to 07/31	07/31 to 08/07
Counting available wine	🟦	🟦	🟦	🟦	🟦			
Quality Control and Tasting	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Placing orders	🟨	🟨	🟨	🟨				
Delivery and restocking				🟪	🟪	🟪	🟪	
Tasks Weeks	06/12 to 06/19	06/19 to 06/26	06/26 to 07/03	07/03 to 07/10	07/10 to 07/17	07/17 to 07/24	07/24 to 07/31	07/31 to 08/07
Current Week								

Figure 1. Simple Web Creation of Gantt Charts

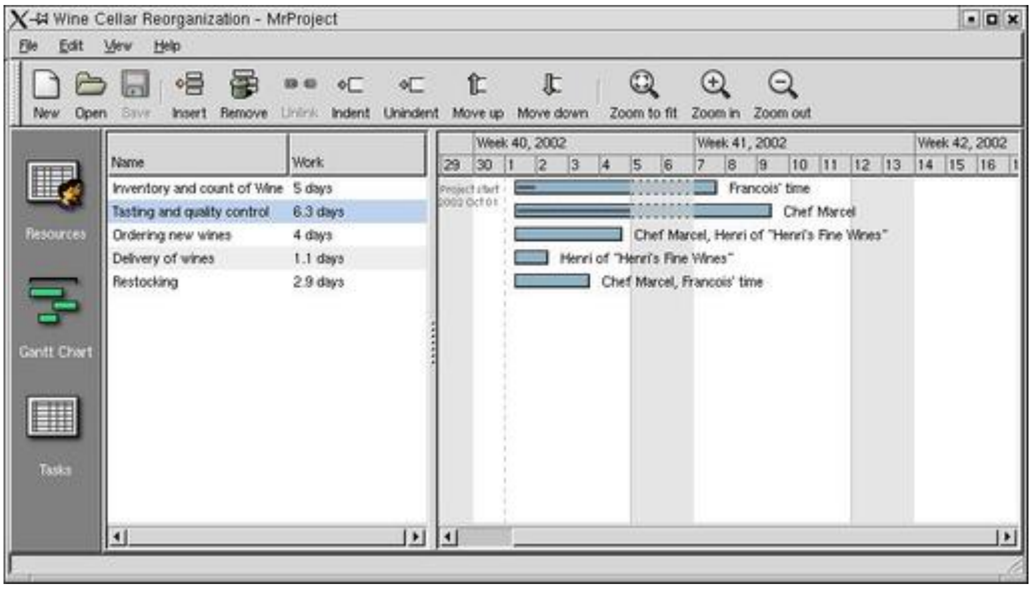
Another project well worth your consideration is MrProject (mrproject.codefactory.se). Part of GNOME Office, this is a desktop-centric application. Consequently, you may not have to do anything if you installed GNOME as your desktop (or even if you did not). MrProject can be found on the distribution CDs of the latest Mandrake, Red Hat, SuSE and others.

When you start MrProject for the first time, by typing **mrproject &**, you open up a blank project. You even can open multiple projects by clicking on New. Notice the buttons on the left-hand side: Resources, Gantt Chart and Tasks. Switching from one view to the other is as simple as clicking the buttons.

When you start your new project, click File on the menubar and select Project Properties. Enter the name of the project, its start date (MrProject has a friendly drop-down calendar to select from), the manager's name and the organization information. Click Close, and save the project under a name that makes sense to you.

Your next step might be to enter the resources you have at your disposal for the duration of the project. You do this by switching to the Resources display and clicking Insert. A default resource record will be added to the list that you can then right-click and edit. A resource might be materials or a person's time. You also can assign cost here.

Under tasks, select Tasks from the sidebar and click Insert once again. As with resources, tasks are generic and need to be edited. These tasks can be described in any way you wish and assigned to one of your resources. Don't worry about the order in which you enter these tasks. You can change the order in the list by selecting a task, then clicking Move up or Move down. The amount of time allocated to a task is entered in days, but you can enter portions of days. The percentage of task completion is also entered here.



Name	Work
Inventory and count of Wine	5 days
Tasting and quality control	6.3 days
Ordering new wines	4 days
Delivery of wines	1.1 days
Restocking	2.9 days

Francois' time
Chef Marcel
Chef Marcel, Henri of "Henri's Fine Wines"
Henri of "Henri's Fine Wines"
Chef Marcel, Francois' time

Figure 2. MrProject's Gantt View

At any point, you can switch to the Gantt view (Figure 2). What's cool is that you can modify the time on the tasks by clicking on the horizontal bar and simply dragging it. (I think I need to allocate more time under “Wine tasting and quality control”). Task dependencies also can be added at any time. After all, some tasks require the completion of other tasks before they can be started.

These two are but a few of the packages designed to handle project management, tracking and charting. If you want to see at some of the other available offerings out there, I would highly recommend a visit to the “Call Center, Bug Tracking and Project Management Tools for Linux” page (www.linas.org/linux/pm.html).

Scroll down to the Project Management as well as the Schedulers, Planners and Gantt Chart Tools section for additional packages to explore. Offerings come from both the freeware and commercial software worlds, and each package comes with a brief description along with links to the package's home page.

Looking at the clock, *mes amis*, it seems as though closing time is nearly here. Perhaps we can take on some enterprise to extend the days by a few hours. With the skills of all the open-source programmers of the Linux world, certainly anything is possible. It is a bold idea, but is that not the meaning of *enterprise*? At the very least, we shall be able to chart our progress, *non*?

Speaking of progress, I see your glasses are nearly empty. Let's ask François to rectify this right away. Until next month *mes amis*, let us all drink to one another's health. *A votre santé*! *Bon appétit*!

Resources

Marcel Gagné lives in Mississauga, Ontario. He is the author of Linux System Administration: A User's Guide (ISBN 0-201-71934-7), published by Addison-Wesley (and is currently at work on his next book). He can be reached via e-mail at mggagne@salmar.com.

An Introduction to FreeS/WAN, Part II

Mick Bauer

Issue #106, February 2003

Connect two private LANs securely with a FreeS/WAN tunnel that runs on your existing firewall systems.

Last month I introduced FreeS/WAN, Linux's implementation of the IPSec tunneling protocol for secure virtual private networks (VPNs). For my sample configuration, I used the common scenario of remote access (RA) VPN. RA VPNs, you'll recall, are used when each remote user is expected to connect to the home network using separate connections, resulting in a one-tunnel-per-user setup.

But what happens when some or all of your remote users are connected to the same local area network (LAN)? I mentioned this type of site-to-site VPN scenario last month, but I didn't explain how to set up one. Building site-to-site VPNs with FreeS/WAN, therefore, is our focus this month.

Architecture: Site-to-Site VPNs

Before we dive into FreeS/WAN configurations, let's take a quick look at architectural considerations. Figure 1 shows a typical site-to-site VPN network layout.

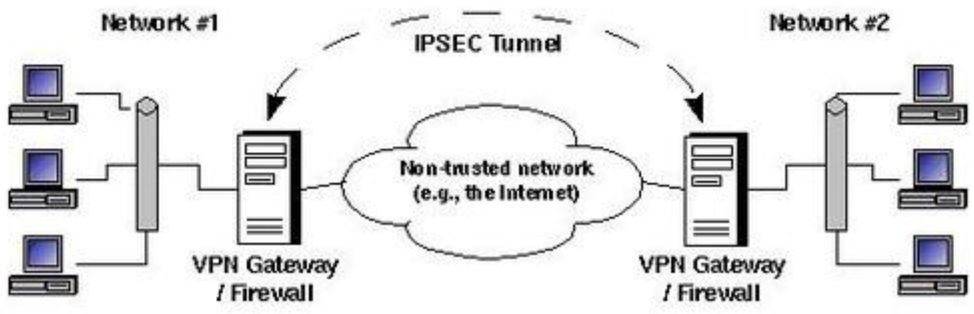


Figure 1. Simple Site-to-Site VPN Design

In Figure 1, each site's firewall acts as a tunnel endpoint. There are several good reasons to use a firewall as a VPN endpoint:

1. Convenience: most firewall platforms support IPSec or some other VPN protocol, eliminating the expense and time required to configure and administer separate VPN servers.
2. Security: a firewall acting as a VPN endpoint can regulate traffic entering and leaving VPN tunnels with excellent granularity and accuracy.
3. Simplicity: if your firewall and IPSec software were designed to run together on the same host, it can be much easier to get your tunnels working and to troubleshoot them when they don't.

However, there are several reasons why this type of setup may not be feasible or desirable:

1. Non-interoperability: if you aren't in control of both sides of the VPN tunnel (e.g., if you're connecting to a vendor's or partner's network), the remote firewall's VPN implementation may not be compatible with your firewall's.
2. Performance: if your firewall is already fully or over-subscribed doing its normal duties, it may not be able to support the added overhead of VPN authentication and encryption.

If, for these or other reasons, you can't use your firewall as a VPN endpoint, you may prefer to use an architecture such as the one in Figure 2.

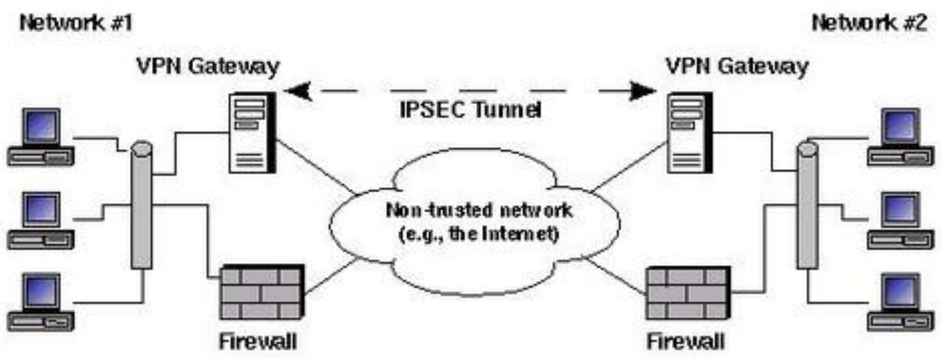


Figure 2. Alternative Site-to-Site VPN Design

In Figure 2, each VPN endpoint is a dedicated computer (in Figure 2 *both* endpoints are set up this way, but you can also mix and match, say, a combined firewall/VPN endpoint on one end and a split on the other). It may seem reckless to put any device in parallel with your firewall. Couldn't such a device be used as a back door?

Indeed, it could—unless the VPN server is carefully configured to accept *only* VPN traffic and its VPN software is carefully configured to accept VPN connections from only approved endpoints, i.e., using strong authentication mechanisms.

Let's jump right into FreeS/WAN and see how to set up a site-to-site VPN with endpoints secure enough to reside either on firewalls or on standalone hosts.

An Example Scenario

Figure 3 shows a site-to-site VPN scenario that's functionally equivalent to the one in Figure 1. That is, it also has the same host at each site serving as a combined Linux firewall and FreeS/WAN IPSec server. Figure 3, however, offers a bit more detail. First, you can see that each network is connected to the Internet via a local router. Second, Figure 3 shows the IP addresses needed for tunnel definitions (we'll see which IPs get used where shortly).

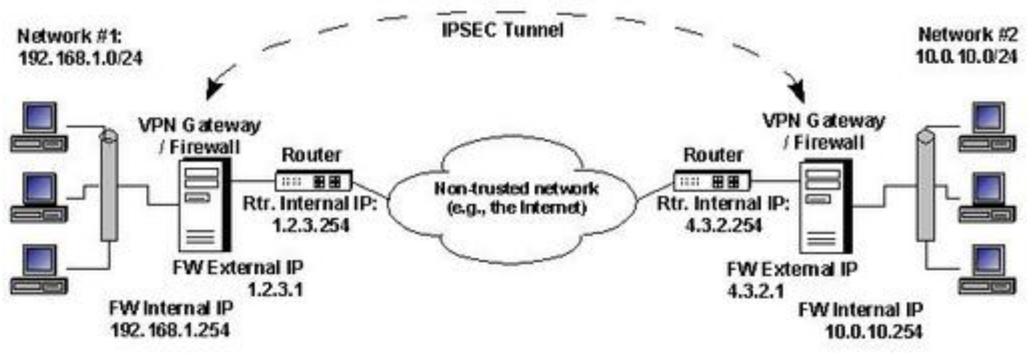


Figure 3. Our Example Site-to-Site VPN Scenario

In this scenario, we need to set up a VPN tunnel between two sites' firewalls' respective “external” interfaces. When a user on one site's LAN wishes to communicate with a host on the other LAN, the firewall sends those packets through the tunnel. Reply packets take the same path back through the tunnel. Hosts on either side may initiate connections through the tunnel.

The firewalls restrict what sort of data may enter and leave the tunnel at either side. On a combined iptables/FreeS/WAN server, these firewall rules can be the same, as though no tunnel were being used, even if network address translation (NAT) is involved. This point is explained later in this article.

A few important premises about this scenario should be noted. First, both firewalls are running Linux kernel version 2.4.18. Second, both firewalls' kernels have been patched with FreeS/WAN version 1.97 and had the user-space FreeS/WAN tools (same version) installed as well. Third, the two networks can reach each other *without* IPSec, i.e., in the clear. (We don't *want* them to communicate that way, but we need to know they *could*; otherwise troubleshooting VPN problems are much harder.)

Exchanging Host Keys

Another ground rule for this scenario is using RSA authentication rather than a “shared secret”. While I don't want to completely re-explain last month's material on host key generation and maintenance, it's important to review the most important points.

You hopefully recall that each host running FreeS/WAN should have a unique host key; you should *not* use the default key provided by the FreeS/WAN binary package you installed. Once you've generated a new key on a given host, however, you'll be able to use that key for as many different tunnels as that host needs. The key remains useful for as long as the secret portion of the host key (stored in /etc/ipsec.secrets) is kept hidden or until advances in cryptanalysis render your host key too short. (Actually, the chances of this occurring before FreeS/WAN itself becomes obsolete for some reason are pretty slim.)

To generate a new host key using FreeS/WAN 1.92 or higher, enter this command:

```
# ipsec newhostkey --hostname my.host.fqdn \
--output /etc/ipsec.secrets --bits 2192
```

This generates a 2192-bit RSA key, saving both its public and private components in /etc/ipsec.secrets. I didn't point out last month that because these commands deal with RSA keys, longer key lengths are required than for, say, a block cipher such as 3DES.

Do *not* be tempted, therefore, to use a value of 128, 196 or some other three-figure number for newhostkey --bits. Public key mechanisms such as RSA and DSA work differently, and their keys must be roughly ten times longer than block- and stream-ciphers' keys. 1,096 bits is the smallest RSA key size you should even consider; 2,192 is much safer.

To display your new public key in a format that can be directly copied and pasted into tunnel definitions, use this command:

```
bash-# ipsec showhostkey --left
```

You can use the option --right instead if you want to print a rightrsasigkey statement instead of a leftrsakey statement.

Remember, the output of this command may be shared safely. It contains only the public component of your host's signing key. You may e-mail it without encryption, post it on a web site or set it to music and sing about it at your favorite coffee shop. This is why RSA authentication is more convenient than shared-secret authentication, in which you *must* securely and covertly send the authentication credentials (shared-secret string) to another site any time you wish to build an IPSec tunnel. RSA authentication allows you to be sloppy (except with /etc/ipsec.secrets, which must be kept root-readable-only at all times); shared-secret authentication does not.

Setting Up ipsec.conf

FreeS/WAN's main configuration file, other than /etc/ipsec.secrets, is /etc/ipsec.conf. In the interest of simplifying things, FreeS/WAN was designed in such a way that tunnel definitions usually look the same on both endpoints of a FreeS/WAN tunnel. Most of the example lines that follow, therefore, are the same on both firewalls in our example scenario.

Last month I focused mainly on tunnel definitions. We'll get to them here, too. But first, let's delve a little deeper into the config setup and conn %default sections. Listing 1 shows a config setup for one of our firewalls (it doesn't matter which one).

Listing 1. Basic Setup in /etc/ipsec.conf

The first parameter in Listing 1, interfaces, is crucial. It defines the interface on which the host will listen for IPSec connections from other IPSec servers. This is not to be confused with the interface on which the host listens for packets *sent through* the tunnel. If you think of the Internet (or other untrusted network) as the outside and the local LAN as the inside, always make sure that the interfaces' parameter is set to your outside interface.

The two debug options, klipsdebug and plutodebug, determine how much logging is done by FreeS/WAN's kernel-interface dæmon (KLIPS) and IKE keying dæmon (Pluto), respectively. Both of these parameters accept the self-explanatory magic values all and none, plus a variety of specific IPSec attributes/events that can be logged. See the ipsec_klipsdebug(8) and ipse_pluto(8) man pages for complete lists of these.

The parameter plutoload specifies which tunnel definitions to initialize when FreeS/WAN starts up. The magic value %search tells Pluto to check each subsequent tunnel definition's auto parameter to determine this (i.e., each tunnel for which auto is set to add).

Similarly, the value plutostart tells Pluto which tunnels to try to connect to automatically when FreeS/WAN starts. In other words, whereas plutoload merely tells Pluto to allow other hosts to bring up specified tunnels, plutostart tells Pluto itself to bring up specified tunnels, without waiting for their other endpoints. Again, the %search value may be specified. In this case, it will match tunnel definitions in which auto is set to start.

Listing 2. Tunnel Defaults in /etc/ipsec.conf

Listing 2 shows the subsequent conn %default section in an ipsec.conf file. The first parameter in Listing 2, keyingtries, is set to zero, which actually translates to no limit. This means when Pluto tries to bring up or replace a tunnel, it tries to key it as many times as necessary. This is a reasonable setting for a site-to-site VPN in which both hosts have persistent network connections, but it's not for a remote-access VPN in which remote clients will be on-line only sporadically.

disablearrivalcheck, if set to no, causes KLIPS to make sure that each packet entering the host from an IPSec tunnel has plausible source- and destination-IP addresses in its header. The default value is yes, which prevents these checks, but you should set it to no unless you really know what you're doing.

Finally, authby lets you choose the default authentication method for tunnels, which, as I said earlier, will be via RSA (rsasig) for our example scenario. And now we arrive at our actual tunnel definition—it's displayed in Listing 3.

Listing 3. Tunnel Definition in /etc/ipsec.conf

Because this is a site-to-site scenario, FreeS/WAN's convention of server = left, remote-access clients = right isn't meaningful. So it's completely arbitrary which side is designated right or left. The important thing is to be consistent across the tunnel definitions in *both* hosts' setups. Here, the site to the left of the Internet (Figure 3) is left, and the site to the right of the Internet is right. That sounds obvious, but if I were to decide to make right left and vice versa, *the tunnel would behave the same* (provided I used the same configuration on both sides).

As you can see, in Listing 3, left is set to the external (Internet-reachable, tunnel-listening) interface's IP address. leftsubnet, however, is set to the address of the network that receives incoming packets (i.e., leaving the tunnel).

leftnexthop is the IP address of the next hop between the firewall/IPSec host and the Internet. And leftrsasigkey obviously is the host key of left. This line (and the comment above it) can be obtained verbatim by running the command **ipsec showhostkey --left**.

The right parameters are the same, but for the right side. I leave it to you to use your powers of deduction to figure out to which hosts in Figure 3 these parameters correspond.

Finally, we have the tunnel's auto parameter, which is set to start. When the Pluto dæmon executes its search for instructions on what to do with tunnel definitions at startup (as described in the section following Listing 1), this setting tells it to initiate the tunnel defined above.

As I've been hinting, in this example scenario, the /etc/ipsec.conf files for both firewalls and gateways are identical. Once they're set up, we can start IPSec on each host and start tunneling. The command to do this on most distributions is:

```
bash-# /etc/init.d/ipsec start
```

If IPSec is already running, use:

```
bash-# /etc/init.d/ipsec restart
```

Once IPSec has been (re)started on both hosts, the tunnel will come up, and each gateway will begin routing traffic addressed to the other network through the tunnel. This routing is done automatically, based on the leftsubnet and rightsubnet parameters defined in your tunnel definition in /etc/ipsec.conf.

Firewalls and NAT

Naturally, you'll want to restrict what sorts of things hosts from the other network may do on your network and vice versa. I stated earlier that firewall rules on a Linux host running FreeS/WAN aren't necessarily any different from when they are without tunneling. This even holds true with NAT. When writing your firewall rules on each gateway, set up FORWARD, POSTROUTING and PREROUTING rules the same as if you weren't using IPSec—just be careful about interfaces. If you use -i and -o parameters, don't say “eth0” if you mean “ipsec0” (or “ipsec+” if you mean “all tunnel interfaces”). When in doubt, try to stick to IP addresses rather than interface names in your firewall rules.

In addition, make sure that no NAT is performed on tunneled packets. IPSec packets' headers are checksummed in the body of each packet's data field. Rewriting the IP header (e.g., by translating source or destination IPs) violates this message-digest, and weirdness will ensue. You can do NAT on packets as they leave the tunnel or before they enter it, but not while they're in the process.

Whatever else you do, you will need at least three new rules on each gateway to allow IPSec key negotiation and tunneling. In the INPUT and OUTPUT chains, you'll need to permit packets sent to UDP port 500, IP protocol 50 packets and

IP protocol 51 packets. The relevant rules on both gateways would look like what is shown in Listing 4.

Listing 4. iptables Rules to Allow IPSec

Conclusion

With that, you're ready to connect your network securely and cheaply to those of your vendors, partners and acquaintances. Good luck!

Resources

Mick Bauer (mick@visi.com) is a network security consultant for Upstream Solutions, Inc., based in Minneapolis, Minnesota. He is the author of the upcoming O'Reilly book Building Secure Servers with Linux, composer of the “Network Engineering Polka” and a proud parent (of children).

Archive Index Issue Table of Contents

Advanced search

Caring Less

Doc Searls

Issue #106, February 2003

If Linus cared more about what happens outside the kernel, it might be a less useful operating system.

On October 2002's Linux Lunacy Geek Cruise, over a hundred Linux faithful got to hang with Linus Torvalds himself for a week. Although conversation seemed to run to children more often than to technology (between us we have four kids under six), Linus talked enough techno trash for me to gather that the man's mantra consists of three short words: I don't care.

There's a lot of stuff Linus doesn't care about: anything in “user space”, other operating systems, new noncommodity microprocessors, fights over development methods, the whole “free vs. open” thing, etc. The list goes on forever, as there's a vast world of technical and political stuff going on outside the one thing he does care about: the kernel.

Linus opened his talk on the boat with this disclaimer: “I only do kernel stuff. I did user-level stuff about ten years ago—only because without it the kernel isn't usable. I don't know what happens outside the kernel, and I don't much care. What happens inside the kernel I care about.”

By not caring, Linus doesn't mean he has no opinions. Like the rest of us, he has plenty of those. Unlike the rest of us, however, he's a Major Figure whose opinions are given a great deal of weight—even when he goes out of his way to remove gravity by disclaiming any interest in a subject. That's why Linus made news with a number of zero-gravity opinions he offered in his talk on the boat, such as why he doesn't like Intel's Itanium or Apple's Mac OS X.

It's only natural to assume that strong feelings accompany strong opinions; but this seems to be less true for Linus than it is for most people, because he often goes out of his way to explain how little he cares about stuff that might be interesting but also distracting. Politics is a perfect example. In that same talk on the boat, he said, “To me all the politics is just amusement value. I don't care.” And because he doesn't care, Linux is a huge success. In fact, it's getting huger every day.

Another geek cruiser was Roland Smith, director of Global IT Operations at LSI Logic. He explained to me how LSI Logic was gradually converting to Linux pretty much across the board, from fancy engineering workstations to desktops. For them, it wasn't only that Linux is cheap and useful, but that it uncomplicates many things.

Roland's story is consistent with a shift in the direction of news about Linux that I began to sense around the time of the cruise. The OS was suddenly being taken seriously and not simply as a “threat” to Microsoft. It was becoming established as a mainstream OS—perhaps *the* mainstream OS—and the reasons were purely practical. Linux is cheap and easy to deploy. It's about as simple and useful as an operating system can be. These virtues are old hat for the Linux community, but they're new hat for many of the world's suits, who aren't used to an OS that doesn't obsolete itself as a matter of policy.

One of the panel discussions on the boat raised the subject of obsolescence. Linus pointed out that commercial software is based to some degree on a model that values obsolescence. Case in point: when Windows 98 came out, Bill Gates was asked about threats from Mac OS. Gates dismissed the question and said the real enemy of Windows 98 was Windows 95. His goal, plainly, was to extract fresh revenues from his entire customer base. Apple clearly has similar plans with major new releases of Mac OS X. For two decades, customers have taken this imperative for granted. They had no choice.

Linux lets customers choose an OS that doesn't care to obsolete itself. That choice became much more interesting to a lot of suits last summer when Microsoft raised its licensing rates for Windows. In a down economy, this rate increase put customers in a much better frame of mind to entertain the Linux alternative.

At the kernel level, Linux doesn't have a commercial agenda. Its purpose is to be useful, period. If you can find a way to make money with it, fine. Linus and his kernel don't care. Sure, some things do get obsoleted along the way. In his talk, Linus explained how the 2.6 kernel will have a whole new block device layer. But even those changes are not being made for the sake of obsoleting anything. They're being made so the kernel will be more useful, in more ways, for as long as possible.

The inherent practicality of the Linux kernel extends upward through the countless choices it supports. By contrast, it's hard to imagine Microsoft or

Apple wanting to support multiple desktops or UIs by developers other than themselves. But that's exactly what Linux does. It supports those desktops and UIs by not caring about them.

Dave Sifry explains how it all works:

> By focusing on a strong separation between kernel-space code and user-space code, the kernel is more stable and strong user-space projects increase momentum. For example, by reducing the amount of code in the kernel, projects like Samba have been able to innovate in a decentralized manner and to create more stable, feature-filled code. No one has to post a patch to linux-kernel in order to get changes made to Samba. All of the major subsystems of Linux share this attribute—XFree86, GNOME and KDE, browsers, heck, even glibc (although the glibc example isn't as strong as the others). We also don't have to worry about Linus creating hidden APIs to make OpenOffice better than AbiWord, or Mozilla better than Opera or KDE better than GNOME, no matter which of those Linus personally prefers.

Not caring is the ultimate level playing field, and it tends to best support other level paying fields built on top of it. For example, while Debian is perhaps the least commercial of all the major Linux distributions, it has provided extremely practical commercial “solution” building material for the likes of Lindows and Xandros. If Debian were busy caring about those implementations, it might be a lot less practical.

“Transparency” is another old-hat Linux virtue that's becoming a hot buzzword in the world of suits. How long before customers demand the same absence of opacity in their OSes as they do in their accounting systems? Hey, why care? It's going to happen anyway.

Doc Searls is senior editor of *Linux Journal*.

Don't Code for Linux

Haavard Nord

Issue #106, February 2003

Better to pick the platform after you write the application.

The days of developing applications for a single platform are history. Why? Because every platform offers at least one key benefit that cannot be attained on any other; Windows, Linux/UNIX, Mac OS X, embedded Linux and others each offer unique advantages. But given changing market conditions, it's impossible to predict which platform will give you the competitive edge you need.

Our anwser is: why pick? We feel that developers can and should leverage all the best qualities of each platform by embracing multiplatform development. This is true not only on the desktop, but also on the server, the network, mobile devices and every other tool that connects us. Our increasingly mobile working style demands portable data and portable applications to match today's distributed networks and global organizations.

Organizations that want to compete and survive must recognize a fragmented OS environment as a given, and they must respond by developing applications that run quickly, cleanly and natively on the greatest number of platforms possible. Applications written in this way take advantage of the best features each platform has to offer, without having to be written and rewritten for every instance. This process limits your company and represents a colossal waste of time. Forward-looking companies already recognize that single-platform development is destined to fail, and they have embraced a better way. Here's why we think this report on the death of single-platform development is not exaggerated.

Single-Platform Development Is Expensive

If you wish to develop for more than one platform, thus expanding your target market, your costs rise dramatically. You need a full team to develop for each platform. Perhaps more importantly, you need a full team to maintain and support each platform. This represents a linear increase in cost for each platform—an extremely inefficient way to do business.

Single-Platform Development Closes Doors

Developing applications for one platform increases your risk because you have to choose among markets before their potential is clear. Who's to say you'll be right? Software companies have been made or broken by this choice. In the recent past, people said Windows (with its momentum and market dominance) was the obvious choice—but wait! Linux has proven itself as a serious competitor in the server space and is picking up serious momentum on the desktop and in the embedded space. World-class consumer and enterprise companies are embracing its power, flexibility, security and low cost. So, what used to be an obvious platform decision isn't so obvious anymore. Do you know when (or where) this kind of rapid transformation will happen again? I don't.

Single-Platform Development Does Not Encourage Mobile Deployment

Perhaps most importantly, if you limit development to a single desktop or server platform, you immediately restrict your access to the fastest-growing software market in the world: mobile systems. If, for example, you write an application for Microsoft Windows NT/2000, you automatically eliminate any cost-effective way to run your application on a mobile device, because you have to rewrite the source. Given that it's nearly essential to make applications mobile, developing applications on a single desktop/server platform can be a death sentence for that application even before it's finished.

The software industry has struggled for some time to develop commercially viable strategies for multiplatform development, and its history is littered with companies that have tried to do this, and failed. Why?

One difficulty has been a lack of complete functionality. Many toolkits deliver only subsets of functionality on multiple platforms, not the whole set. Another problem has been reliance on emulation or virtual machines. Both of these impose a significant and usually unacceptable performance penalty, especially for mobile devices that need high performance the most.

It's a well-recognized fact that differences between virtual machines lead to implementation workarounds and tweaks, as well as increased maintenance. This is another expense, and it makes the developers who have to do this work miserable.

Today, though, proven ways exist to write an application once, compile and run it anywhere. Companies who do multiplatform development create an environment in which development innovation will once again be the order of the day—not the exception.

Haavard Nord, cofounder and CEO of Trolltech, started his programming career trying to find acceptable multiplatform toolkits for database development. He now drives Trolltech's efforts in single-source, multiplatform software development. The company's products encourage innovation by letting developers write single-source applications that run natively on Windows, Linux, UNIX, Mac OS X and embedded Linux.

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #106, February 2003

Readers sound off.

Cluetrain Station: ASA

The fact that a user-friendly distro such as Mandrake tops your poll tells me that readers are interested in the practical uses of the OS as a day-to-day computing platform. Could more non-tech reviews be in order? I recently went shopping for a Linux machine. The “Dell Dude” didn't want to talk to me unless I was willing to give some money to a guy in Redmond; however, from your pages I contacted the nice folks at ASA Computers. Their agent Sean acted as if I were a major buyer, and in no time I had a nice, new Linux workstation delivered to my door.

—Michael Presley

Please Run Microsoft Ads!

In the November 2002 issue [Letters, page 6], Renato Carrara told you not to run Microsoft ads. I present you the top five reasons *to* run MS ads in *LJ*: 1) Bill Gates is NOT evil. 2) MS ads in *LJ* would be proof that we won the OS battle! We all remember when MS ignored us, claiming that Linux had too small of a user base to be interesting. 3) Let *LJ* readers, not staff, make the decision on whether the offer is hot or not. 4) If MS pays money to *LJ* for bringing an ad, we, the *LJ* readers, end up with a better magazine! 5) Linux is not just about having access to source code. It is also about freedom—freedom of choice, free markets and free competition. MS ads cannot disillusion, scare or harm us! They can only make us stronger!

—Martin A. BoegelundBjerringbro, Denmark

Memory Leaks No More

I just read the article about memory leak detection in the September 2002 issue of *LJ*. I was surprised to see that Valgrind, which blows all three mentioned memory checkers out of the water, was not mentioned. Valgrind (developer.kde.org/~sewardj) is every bit as good as purify from Rational Software, which is not available for Linux, or Insure++, which is very expensive.

—Greg Hosler

USA! USA!

It's tough getting a vanity plate done in Connecticut; we're very conservative. I was surprised no one had taken the Linux plate, so I had it made with the new patriotic theme that our DMV just released. I think it makes a positive statement. Yes, Tux is helping me show it off.

—Paul Ammann

Sorry, Tim

Thanks for printing the Henson Control System article [“Controlling Creatures with Linux”, *LJ* November 2002]; it came out quite well, I think. My apologies to Tim Magill, whose name I misspelled by trusting spell checking.

—Steve Rosenbluth

We Are the World

Reading the November 2002 issue of *Linux Journal*, I have had a feeling about i18n/i10n that I must share with all of you. To be clear, I must say that we Italians were united by wars between 1861 and 1918, but we are still using our pre-unitarian languages, or “dialects”, so I really appreciate the localization effort. But at a global scale, actually the English language is used as a *lingua franca* for the diffusion of scientific and informatics knowledge. If we eliminate such an instrument we'll break the free transmission of ideas. I think that we must have care of both sides of the phenomenon.

—Franco Favento de i Favento de Trieste

Game Articles, Please

Many thanks for putting so much effort into *Linux Journal*, but I do notice something that seems to be missing—a gaming section.

—Matthew

We don't have space for a gaming section each month but will include screenshots and mini-reviews of fun games as we discover them. Please send your favorites to info@linuxjournal.com.

—Editor

Back to Brazil

I received a copy of the article “Free Software in Brazil” by Jon Hall, featured in the September 2002 *LJ*, from a friend who lives in the US and who was quite worried about some things the article mentioned regarding what you call “the Landless Movement” in Brazil.

Although the cause of this movement may look noble, there are many things related to this group that should have been considered before publishing that article and stamping their flag on the magazine's cover. The group, known here in Brazil as MST, has a radical political position, and they are responsible for some actions that could be seen as terrorist acts, such as mass killings and unauthorized land occupation including invading and destroying a farm that belongs to Brazil's president, Mr. Fernado Henrique Cardoso. The group's default way of acting is by the use of brute force and they usually threaten people to get what they want. Everything I'm mentioning here can be easily checked. A simple search on major Brazilian newspapers will show that the MST isn't worth trust or being featured on a magazine such as *Linux Journal*. Mr. Jon Hall, with all due respect, should try to find out better who he is supporting or featuring.

—Bruno Trevisan

Jon “maddog” Hall replies: The group of people that I was introduced to, and who appear in the picture, were introduced to me by Mr. Marcos Mazoni, the President and CEO of PROCERGS. PROCERGS is the government-owned software company that is doing a lot of the open-source programming for the state of Rio Grande do Sul. The group also showed up at the third annual Software Livre! conference, which was sponsored in part by PROCERGS, and attended by the Mayor of Porto Alegre and a representative from UNESCO. There was no mention or indication that this was a “terrorist group”. However, in light of Mr. Trevisan's letter I went back to Brazil and did some more research, both directly and indirectly, by reading and by talking to people who I trust and by doing a little web-scouring of my own.

In Brazil about 60% of the land is owned by 1% of the population. A lot of this land lies idle, while millions of people are out of work and have no way to generate money or food. Since 1984, MST has won land for approximately 250,000 families in 1,600 settlements, but there are still another 4.8 million families without the means to support themselves. MST is building cooperatives, building agro-industries, building and staffing day-care centers, teaching literacy classes to 25,000 adults, helping to educate 150,000 children in 1,200 public schools, and they are proud of their nonviolent direct action in the form of land occupations. I have lived through quite a few other “actions” in my lifetime. The Civil Rights movement of the 1960s and the anti-Vietnam War movement come to mind. Both of these had their peaceful incidents and incidents that were not so peaceful. However, to judge a whole movement and its motives by the actions of a small minority of them is wrong. If the whole movement of the MST is indeed bad, and if they are indeed terrorists, then I have good company in the number of people and agencies that have been duped by them. It appears that UNESCO, the Catholic Church, and even parts of former President Cardoso's own judiciary and elected officials supported the MST.

Actually, from the documentation I found, it seems that the nonpeaceful and “terrorist” actions seems to have come from the former government of Brazil, with the killing of many of the protesters by government forces and the lack of action by the government in holding their killers responsible. One article written by Mark S. Lagevin, a professor of Global Studies at Pacific Western University, says there were 969 assassinations of rural workers and MST activists, with only five people convicted of those crimes. Hopefully, the new government will be more responsive.

Luiz Inacio Lula da Silva, of the Brazilian Worker's Party (PT) is now the President of Brazil. In the past the PT has helped to support the MST in its quest for land reform, and I hope this continues. Not relying on only newspaper articles and web sites, I asked several of my Brazilian friends what they thought of the MST movement. Each of them stated that while they deplore any type of violence, the good that the MST movement generates is many times better than the reported bad actions of a few, and that the past government had not been responsive to more than reasonable requests for land reform. Therefore, they supported the MST overall.

However, I was informed by a friend in São Paulo that representatives of Microsoft were taking copies of *Linux Journal* to people in business and telling them that I was not an honorable person because I was associated with the MST group, and by default if I am not honorable, then the Open Source movement must not be honorable. Therefore, I would like to set the record straight as to what type of person I am. I believe in free speech, a democratic process and freedom of choice. I believe in nonviolence and have never physically hurt anyone in anger. I have never been in jail, never been arrested and have received only two speeding tickets in my entire life (which I paid). I do not believe that killing a human being for anything less than immediate threat of life (my own or another person's) is ever justified, yet I would go to war to protect my country and its ideals if we were attacked. I believe in equality of the races and sexes, and I believe in honoring diversity in religion and sexual orientation. I believe that the duty of government is to honor the will of the majority as long as it does not violate the rights of the minority. I honor laws, even those that I am not completely happy about, believing that it is better to change them than to break them. I encourage people to think about what the law means, and what it would be like if there were no laws. Finally, I have never even been accused in court (local, state or federal) of being a monopoly, breaking the law or knowingly harming another person's business or the consumer by my business tactics. As for my association with the MST group, and my picture alongside their flag, I repeat that my interest was in seeing people who had little money and great ideals using a free and open operating system to help their cause and better their lives. On this path I will continue to walk. With respect to Mr. Trevisan, I have encouraged *Linux Journal* to publish his letter, and this response. For those of you who are interested in the MST movement, I offer their web site at www.mstbrazil.org or www.mst.org.br.

Advanced search

Upfront

Various

Issue #106, February 2003

What's new with the kernel and more...

diff -u: What's New in Kernel Development

Folks waiting for **LVM1** to be fixed in the kernel can stop waiting. It's been removed, after six months of sitting in an unmaintained, broken state. **Joe Thornber** posted the actual patch to take it out. This seems to be part of a general “let's get cleaned up for 2.6” push. There was talk on the mailing list of replacing the code with **LVM2**, **Device Mapper (DM)** or even **EVMS**. None of these were universally hailed as the obvious choice. DM had a lot of missing features, while EVMS had too many available ones. There was even some talk of trying to fix LVM1 instead of pulling it out—or at least of finding a suitable replacement first. So far, EVMS looks like the prime candidate for 2.6, but it's still too soon to say.

At the Ottawa Kernel Summit it was agreed that **driverFS** would be changing its name, and patches have begun appearing to do exactly that. The only problem is no one can agree on what the new name should be. **Patrick Mochel** wants “kfs”, but others say that single-letter-plus-fs is getting to be a cluttered namespace. **H. Peter Anvin** has suggested “kernelfs” or simply “kernfs”. About the only thing that's known so far is the name will change.

Support has been added for the **NEC PC-9800** architecture, a popular architecture in Japan that is roughly the equivalent of the Intel-based PC in the West. Traditionally, it has run ported versions of MS-DOS and Microsoft Windows, although it has never been fully “IBM-compatible”. With 40-50% PC market share in Japan, these patches open Linux up to a huge number of people who previously may not have had access to it.

Jeff Dike's User-Mode Linux now has SMP support. Up until now, regardless of the number of CPUs on your system, UML processes were entirely uniprocessor. Among other things, this meant that testing SMP software (particularly the kernel itself) under UML was not going to get you much. This new advance opens the door for speedier testing of various applications that otherwise would require many lengthy reboots, possibly accompanied by filesystem corruption.

The **ioctl** interface is now deprecated. New drivers should create a filesystem-based interface with **libfs**, newly included in the 2.5 tree. I/O control functions have been condemned for years as an unmaintainable, undocumentable, ever-growing mess, but for a long time there was no way around it. Now at last, Linux developers can use an interface that makes sense—one that won't cause more trouble than it's worth.

The kernel went into **feature-freeze** on October 31, 2002. It's much too soon to tell if this will lead to 2.6 in a reasonable amount of time, or if events will lead back to a period of rapid development with no end in sight. **Linus Torvalds** and others have been struggling for a while to bring the kernel rev-time into a reasonably short time frame, but the time between stable series is still measured in years. If we see 2.6 before April 2003, it will be a major achievement in the development of the development process itself.

—Zack Brown

Think of It as a Hacksess Point

Wi-Fi (802.11 wireless Ethernet) access points have been around for a while. But most of them allow rather limited programmability, especially if you want to make a business out of selling customized ones.

Chipset maker Intersil (www.intersil.com) has walked into this marketplace and taken care of the opportunity problem by introducing a new self-hosting access point reference design called PRISM AP. What makes it so marketable is its operating system: embedded Linux. PRISM AP comes with a whole Linux development environment on which you can run a web server, a DHCP server, DHCP client and SNMP server, among other things.

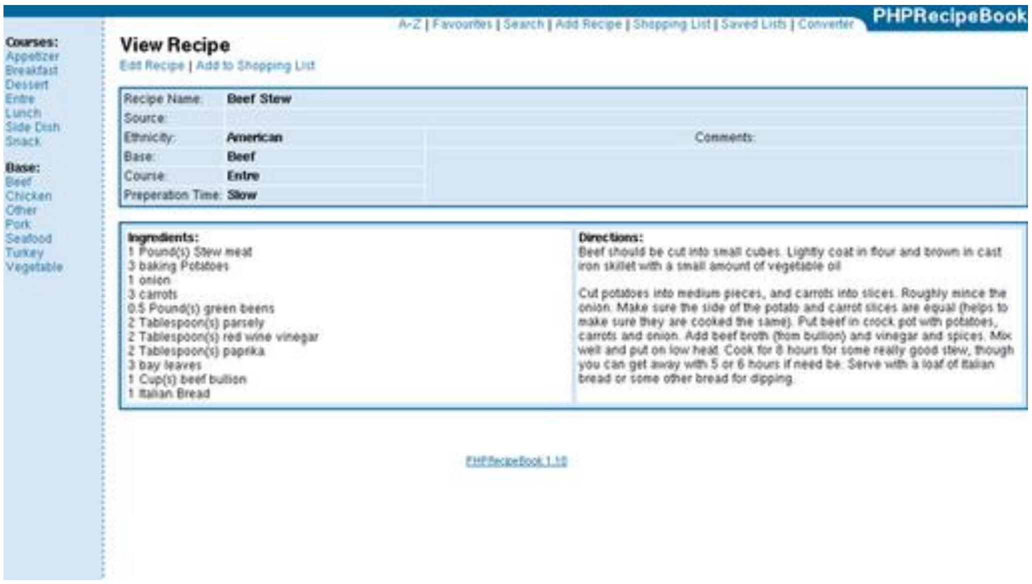
Because the OS is Linux, you can customize units for your own uses or develop whatever product you like—VPN gateway, bridge, router, mesh network, whatever—and burn the code into Flash memory. Then you can sell it without worrying about licensing costs.

In other words, it's hard to imagine anything more equally hackable and marketable, or in more different ways.

—Doc Searls

PHPRecipeBook: phprecipebook.sourceforge.net

If you need to keep a recipe book, this is the ticket. You can input ingredients and the preparation process and save it. When you decide you want to cook something, the program can generate a list of ingredients in a shopping list and you can print it out. Now, if you could just ship it off via the Web to a local grocery store and have the ingredients delivered, you wouldn't even have to leave the house. Unfortunately, recipes are not included. Requires: web server with PHP and SQL (PostgreSQL or MySQL) support, SQL server and a web browser.



View Recipe

Recipe Name:	**Beef Stew**	
Source:		
Ethnicity:	**American**	Comments:
Base:	**Beef**	
Course:	**Entre**	
Preperation Time:	**Slow**	

Ingredients:
1 Pound(s) Stew meat
3 baking Potatoes
1 onion
3 carrots
0.5 Pound(s) green beens
2 Tablespoon(s) parsely
2 Tablespoon(s) red wine vinegar
2 Tablespoon(s) paprika
3 bay leaves
1 Cup(s) beef bullion
1 Italian Bread

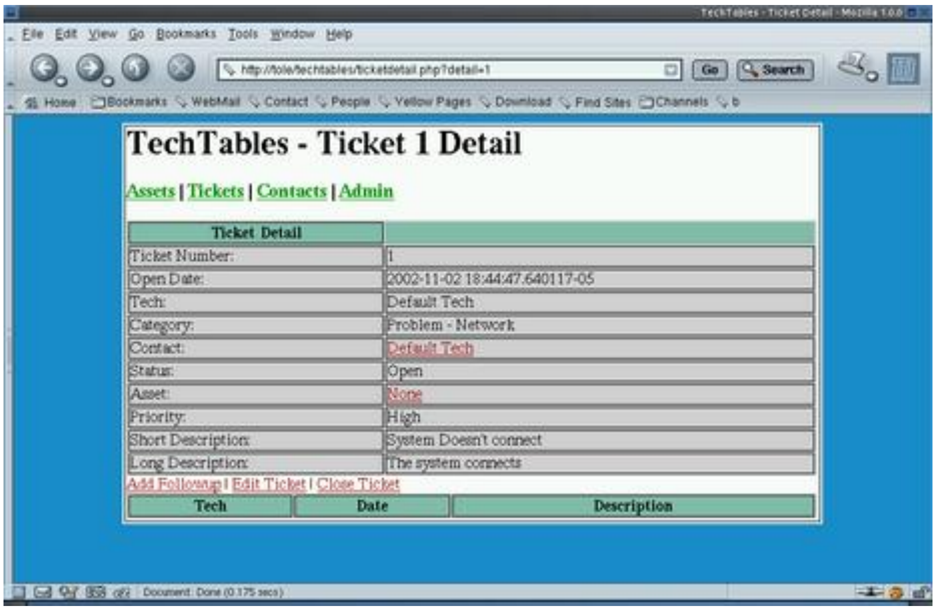
Directions:
Beef should be cut into small cubes. Lightly coat in flour and brown in cast iron skillet with a small amount of vegetable oil

Cut potatoes into medium pieces, and carrots into slices. Roughly mince the onion. Make sure the side of the potato and carrot slices are equal (helps to make sure they are cooked the same). Put beef in crock pot with potatoes, carrots and onion. Add beef broth (from bullion) and vinegar and spices. Mix well and put on low heat. Cook for 8 hours for some really good stew, though you can get away with 5 or 6 hours if need be. Serve with a loaf of Italian bread or some other bread for dipping.

—David A. Bandel

Techtables: techtables.sourceforge.net

This particular trouble ticket and asset tracking system is a little different from some others. It deals fairly exclusively with trouble tickets and assets, not so much with clients and the the rest. Depending on your needs, this might fill the bill nicely. Installation and use is simple. If you need to protect anything, you'll need to implement htpasswords and/or secure web support (easily done). Requires: web server with PHP and SQL (PostgreSQL or MySQL) support, SQL server and a web browser.


TechTables - Ticket Detail - Mozilla 1.0.0
File Edit View Go Bookmarks Tools Window Help
http://tole/techtables/ticketdetail.php?detail=1
Go
Search
Home Bookmarks WebMail Contact People Yellow Pages Download Find Sites Channels

TechTables - Ticket 1 Detail

Assets | Tickets | Contacts | Admin

Ticket Detail	
Ticket Number:	1
Open Date:	2002-11-02 18:44:47.640117-05
Tech:	Default Tech
Category:	Problem - Network
Contact:	Default Tech
Status:	Open
Asset:	None
Priority:	High
Short Description:	System Doesn't connect
Long Description:	The system connects

Add Followup | Edit Ticket | Close Ticket

Tech	Date	Description

Document: Done (0.175 secs)

—David A. Bandel

They Said It

HP is talking about how HP-UX will be able to run Linux applications; so is Sun with Solaris. ISVs are going to be asking themselves, “Why should I bother to develop for a specific UNIX if I can develop for Linux and it will run on almost all UNIX platforms?”

—Dan Kusnetsky

Middleware? I think it's something that sits between something that's useful and something you can understand.

—David Sifry

You need 64-bit support; you need terabyte filesystems. IBM says the Open Source community will tackle that. I don't think so. Somebody's got to build that.

—Jonathan Schwartz, Sun Microsystems

[*See page 44.*]

Archive Index Issue Table of Contents

Advanced search

Advanced search

From the Editor

Don Marti

Issue #106, February 2003

New developments for Linux in the Enterprise.

There are two common points of view about Linux in the so-called enterprise. The first is that Linux is only capable of displacing Microsoft products on cheap low-end servers, and that proprietary UNIXes with their huge 64-bit address spaces and big SMP scalability are safe. The second is that Linux is mainly displacing UNIX, since it's easy to port software from UNIX to Linux, and that Microsoft with its difficult-to-port-from APIs is safe.

Both points of view are wrong. Nothing is safe. On page 44, Linux is running on a new 64-processor NUMA system. And, on page 52, Linux is displacing the nastiest Microsoft server to replace, Exchange—undocumented protocols and all.

In our December 2002 issue, Douglas B. Maxwell wrote about how he beat the graphics performance of a large SGI system with $15,000 worth of PCs. This month, we're celebrating SGI's release of a big Linux box by putting it on our cover. Do we have the world's shortest attention span? Aren't generic PCs taking over everything?

If they are, they're not done yet. If you have a big problem that you haven't figured out how to split into PC-sized chunks, or don't want to take the time to split into PC-sized chunks, the 512GB of memory on the new SGI Altix 3000 seems like just what you need.

One slogan at SGI is “do science, not computer science”. Do big problems the way you know how and get better results now. Of course, this is waving a red flag in front of the commodity cluster faction, and I'm sure we'll soon have plenty of articles pointing out how you can get previously unclusterable work done on a cluster.

The diversity of success stories in this issue makes it clear that any company that tries to compete with Linux in a fair fight will lose. So it's going to be an unfair fight for a while, with the non-Linux vendors pulling shenanigans such as bogus software patents, FUD-based marketing, copy-restricted content, carefully placed “donations” and “campaign contributions”, and who knows what else.

But most of the companies, and most important, the people, who are promoting non-Linux legacy products today are going to be part of the Linux business tomorrow. Since our community will survive and theirs won't, ours has to be able to welcome and do business with them in the future. So we can't engage in the same desperate nonsense they are. All we have is the best software and the truth, and that's plenty.

Finally, the one enterprise that's most important to pulling some people from just getting by up to knowledge and success is the public library. Your local library provides educational materials, entertainment, training and community programs. Now, that important institution's budget won't be wasted on expensive, inflexible proprietary software. The Koha Project is offering library cataloging and search software under the GPL, in the same spirit we have public libraries in the first place. Join your local Friends of the Library and read Pat Eyler's article on page 58.

Don Marti is editor in chief of *Linux Journal* and number eight on pigdog.org's list of “things to say when you're losing a technical argument”.

Cruising the Carribean

Doc Searls

Issue #106, February 2003

If you weren't on the boat, here's what happened on Linux Lunacy II.

Our second annual Geek Cruise, Linux Lunacy II, combined master tech classes and fun shore excursions with a week-long tour of the Western Caribbean. So far, *Linux Journal* has sponsored two of Neil Bauman's Geek Cruises. Next year's cruise will head North, from Seattle up through Alaska's Inside Passage.

On Linux Lunacy II, more than 120 Cruisers boarded Holland America's *ms Maasdam* in Ft. Lauderdale on October 20, 2002, for a loop around the Western Caribbean, stopping at ports of call in Cozumel, the Cayman Islands, Jamaica and the Bahamas. Sessions were held en route, with attendees gathering in meeting rooms, dining rooms, bars and theaters. The formats ranged from keynotes to lectures to master classes to Q&As. Check out the cruise photo gallery at www.linuxjournal.com/article/6420.

The speaker lineup could hardly be more top-drawer for every subject: Linus Torvalds and Ted Ts'o on Linux; Randal Schwartz on Perl; Guido van Rossum on Python and Zope; Steve Oualline on Vi and C++; Dirk Elmendorf on PHP; Greg Haerr on UI programming and embedded Linux; Eric S. Raymond on open source, hacker culture and the Zen of UNIX; Brian Carrier on forensics and Brandon Wiley on ad hoc serverless communities. I did the keynote on "How Linux Got to Be Everywhere While Nobody Was Watching". The slides from the keynote are available on the *Linux Journal* web site at www.linuxjournal.com/article/6421.

The settings were friendly too. Neil provided power strips so cruisers could plug in laptops, projectors and other peripherals. He also arranged with Digital Seas to provide wireless live internet connectivity, handy for checking out URLs and doing other research while following a lecture. Each time I checked it, throughput ran around 150-250KB downstream and 85-120KB upstream. Not bad for a boat connected to the Net by a satellite 23,500 miles above the Equator.

Greg Haerr attended Ted's “More than You Ever Want to Know about Filesystems” talk, and he said it, “expanded my understanding of the key details in the design and implementation of the 2.5 kernel. In conjunction with Linus' talk, I left with insight into how the kernel guys go about making Linux superior and a better understanding of the way it's managed.”

Linus spoke three times: once in a long Q&A and two more times on panels. My ideas about the future of Linux were confirmed by Linus' talks along with other speakers' presentations on the ship. To see what we're predicting, read “What I Learned on Linux Lunacy” at www.linuxjournal.com/article/6414.

One of Linus' panels met onshore, in Ocho Rios, Jamaica, with the Jamaica Linux Users Group (JaLUG). The setting was The Ruins Pub & Internet Café, which is set back in a jungle of palms and banyan trees beside a perfect waterfall.

That meeting took place early in the morning, after which most of us headed to Dunn's River Falls, which cascades for hundreds of yards over limestone boulders on its way to the sea. Dunn's is an interactive river: the idea is to climb it upstream from bottom to top. It's a little scary but a lot of fun. We probably set a new record for the number of otherwise smart people carrying digital cameras up the middle of a cascading jungle river. For a full description of all the events, see parts 1 and 2 of “Geeks on the Half Shell 2.0: Cruising the New Dominion with Linus and Friends” (www.linuxjournal.com/article/6419 and www.linuxjournal.com/article/6422, respectively).

For the next cruise, we're moving the schedule up to September 13, 2003. We sail out of Seattle for Alaska's Inside Passage aboard Holland America's flagship, the *Amsterdam*. “It's an incredible ship”, Neil says—and he ought to know. The itinerary includes Victoria, Juneau, Sitka, Ketchikan and Hubbard Glacier.

Last year our family went on another of Neil's cruises to Alaska. The sights run from stunning to spectacular. The whole trip flanks jagged snowcapped mountains feeding the sea with rivers, streams and some of the most spectacular glaciers in the world. There's nothing quite like floating next to the business end of a glacier for half a day, watching huge hunks of it “calve” off into icebergs the size of buildings. And that's just what you do on the boat. The shore excursions include helicopter trips, dog sledding...all kinds of fun stuff.

Commitments so far suggest there's a good chance Neil will sell out early this year. Visit the Geek Cruises site www.geekcruises.com to find out more.

Doc Searls (info@linuxjournal.com) is senior editor of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #106, February 2003

Our experts answer your technical questions.

Where's My Printer?

I would like to know how to get OpenOffice to use the system printers. I did a normal install of SuSE 8.0 Pro and set up an Epson Stylus Photo 1280. When I installed OpenOffice, the only printer it set up was generic.

—Nathan M. Fowler Jr., nfowler1@bellsouth.net

You may need to run the spadmin program, which is located in the ~/OpenOffice.org1.0.1/program directory. This is the printer configuration utility of OpenOffice.

—Felipe E. Barousse Boué, fbarousse@piensa.com

Where's My IP Address?

I have a Linksys router. Red Hat Linux detects the Ethernet port built onto my motherboard, but I can't for the life of me get it to connect.

—Tim Kuder, tim@kuderized.com

Try Red Hat's netconfig tool. You may need to set DHCP to get the IP address from a DHCP server instead of declaring one yourself. Then, ping your router's address; if it works, you are set.

—Felipe E. Barousse Boué, fbarousse@piensa.com

Can I Reboot?

I am in the process of installing Red Hat Linux 7 (using the distribution packaged in the *Red Hat Linux 7 for Dummies* book). My installation appears to have hung after installing 153MB of a 753MB installation. Is it safe to reboot my computer?

—Marcus, marcus@lesniak.co.uk

I would hope that your computer isn't still waiting for a response to this request. Yes, it is safe to reboot. However, Linux was not successfully installed, so you will not be able to boot Linux. It's possible the CD you are installing from has some corruption.

—Christopher Wingert, cwingert@cwingert-mail.qualcomm.com

If everything hung up, you probably have no choice but to reboot. Are your installation CDs unscratched? Are you positive your hard disk is in good condition? Re-install from scratch, and this time select the Check for Bad Blocks option when creating partitions. This will take longer, but it will test your disk against some defects.

—Felipe E. Barousse Boué, fbarousse@piensa.com

Can't Boot from CD or Floppy

I have a Sony Vaio PCG F340, and I want to install Slackware on it. Following some bad advice, I formatted the c: drive. Now anything I put in the CD drive or floppy drive comes up with an **invalid system disk** error.

—John Krissinger, kriskrosx@aol.com

Go into your BIOS and change the boot order of your drives. Put the drive you are installing from, either the floppy or the CD-ROM, ahead of your hard drive.

—Christopher Wingert, cwingert@cwingert-mail.qualcomm.com

I'm not certain Slackware can be installed on your Vaio; those laptops are a bit hard to support. If you don't get anywhere with it, you may want to try installing Red Hat, SuSE or Linux-Mandrake. These all have installers that are better at probing laptop hardware.

—Marc Merlin, marc_bts@google.com

Support for NVIDIA Laptop Video?

Please advise me about installing any version of Linux that can work with my Dell Inspiron NoteBook computer. It has a UXGA monitor (1600 × 1200 native resolution) and an NVIDIA GeForce 2 Go video graphics system. Neither Dell nor Red Hat are of any help. I tried to install earlier versions of Caldera and Mandrake 9.0, also without success.

—Kamalakar Rao, kmlkr@juno.com

Your main problem is that the good GeForce drivers have to be downloaded from NVIDIA, because they are not open source. What you want to do is install your distribution of choice with XFree 4.1 or 4.2. Then go to the NVIDIA web site, download the closed-source drivers, and follow the install directions: www.nvidia.com/view.asp?IO=linux_display_1.0-3123.

—Marc Merlin, marc_bts@google.com

How to Set Up DNS and DHCP?

I am trying to set up a Red Hat 7.3 box as a server for my school. I have been looking for advice about how to set up a DNS and DHCP server.

—Richard Whiteside, Hendel4ever@hotmail.com

Red Hat has some nice tools for configuration of the services you require. Read the Installation Guide, which is actually quite good. www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/install-guide.

—Christopher Wingert, cwingert@cwingert-mail.qualcomm.com

You can find excellent tips within the DNS-HOWTO and DHCP-HOWTO. You can find those and many others in your distribution CD or on-line at tldp.org or on many other mirrors.

—Mario Bittencourt, mneto@argo.com.br

Archive Index Issue Table of Contents

Advanced search

New Products

Heather Mead

Issue #106, February 2003

ACCPAC Advantage Series 5.0, ProStore Backup Appliance, Zaurus SL-5600 and more.

ACCPAC Advantage Series 5.0

ACCPAC Advantage Series Enterprise Edition 5.0 is a multitiered, web-based business management system that provides access to your entire accounting system using either a browser or the ACCPAC desktop. Enterprise Edition includes the following functions: system manager, general ledger, accounts payable, accounts receivable, inventory control, order entry, purchase orders and US and Canadian payroll. The general ledger consolidations and intercompany transactions modules also are available. Enterprise edition provides multicurrency and multilingual support; supports unlimited users; and offers compatibility with Oracle, IBM DB2 and Pervasive database compatibility.

Contact ACCPAC, 6700 Koll Center Parkway, Third Floor, Pleasanton, California 94566, 925-461-2625, www.accpac.com.

ProStore Backup Appliance

ProMicro and Avail Solutions have partnered on the ProStore Backup Appliance, which provides data storage, backup and recovery capabilities. The ProStore appliance is a combination of ProMicro's ProStore NAS server and Avail's Integrity backup software and automated tape library. ProStore comes with 360GB of storage capacity, Intel processors, 10/100 Ethernet ports, three PCI slots and up to six IDE drive bays in a compact 2U form factor. The integrated eight-cartridge tape library transfers up to 640GB of compressed data at 6MB/s. All parameters are user-programmable.

Contact ProMicro Solutions, 12635 Danielson Court #203, Poway, California 92064, 866-776-6427, www.promicro.com; Avail Solutions, 2430 Vineyard

Avenue, Suite 205, Escondido, California 92029, 760-743-7200, www.availsolutions.com.

Zaurus SL-5600

The latest addition to Sharp's Zaurus PDA family is the SL-5600, a business and wireless-capable PDA for enterprise users. The 5600 has a high-resolution, color-reflective QVGA LCD screen, a QWERTY keyboard, 64MB of protected Flash memory, 32MB of SDRAM, dual expansion with CompactFlash, SD/MMC card slots and an integrated speaker and microphone. The Qtopia-based PDA also features an Intel XScale 400MHz processor with a 100MHz memory interface. A virtual mobile hard drive is also built in to the SL-5600, protecting data, applications and files in Flash memory. The new Zaurus includes drivers for 802.11b wireless LAN adapters, CDPD wireless modems and 10/100 Ethernet cards.

Contact Sharp Zaurus, 201-529-9459, www.myzaurus.com and www.sharpusa.com.

SCO Linux 4.0

In its first release since dropping the Caldera name, the SCO Group announced the availability of SCO Linux 4.0. SCO Linux 4.0 is based on UnitedLinux 1.0, designed for mission-critical business applications. SCO Linux also comes with software, support and services available through more than 16,000 resellers for small- to medium-sized businesses and replicated branch sites. Four editions of SCO Linux 4.0 are available—Base, Classic, Business and Enterprise—each coming with a one-year support plan.

Contact The SCO Group, 355 South 520 West, Suite 100, Lindon, Utah 84042, 801-765-4999, www.sco.com.

Cubix BladeStation

The BladeStation from Cubix Corporation is a dual-Pentium 4 Xeon blade server with up to four PCI-X/PCI slots per blade. BladeStation also has a 533MHz front-side bus and supports up to four SCSI drives within each blade. Up to seven dual-Xeon blades can be housed in a 6U rack, using only 21 inches of rack depth. Each blade features one full-length 64-bit 133/100MHz PCI-X extension slot, up to 8GB of DDR RAM, a 1GB Ethernet port and two 10/100 Ethernet ports. The PowerStation power supply array provides redundant n+1 power.

Contact Cubix Corporation, 2800 Lockheed Way, Carson City, Nevada 89706, 800-829-0550 (toll-free), www.cubix.com.

Frequency Clock: Free Media System

The Frequency Clock: Free Media System is an open-source software system for managing streaming audio and video channels. Users can organize their streaming-media files into dynamic channels that can then be played back using the web-based streaming-media player. Created by Radioqualia, the key features of the Free Media System include a customized streaming media player that can handle different file types, searchable archives and real-time statistical analysis. All streaming media file types, including Windows Media, Real and QuickTime, can be played in a single player, the Frequency Clock Player. In addition, users can customize the Player to fit their design needs rather than fitting the design to the needs of a specific player.

Contact The Frequency Clock, radioqualia@va.com.au, radioqualia.va.com.au/freqclock/central.html.

Dell 1655MC

The 1655MC is Dell's latest entry in the blade server market and has the equivalent of six two-way 1U servers in a 3U blade enclosure. The 1655MC, which looks like a thick blade in a box, supports one or two 1.266GHz Pentium III processors, up to 2GB of SDRAM and one or two Ultra 320 SCSI drives. The chipset is a ServerWorks ServerSet LE30, and the 1655MC also has two integrated Broadcom Gigabit Ethernet interfaces and a USB port. Dell's chassis, which accommodates six 1655MCs, has two hot-plug power supplies for 1+1 redundancy. It includes a built-in KVM switch, plus either one or two managed Ethernet switches. The 1655MC is available with Red Hat 7.3, 8.0 or Advanced Server.

Contact Dell Computer Corporation, One Dell Way, Round Rock, Texas 78682, 800-915-3355 (toll-free), www.dell.com.