Advanced search

Linux Journal Issue #121/May 2004

Automatic rollback for RPM upgrades
LINUX JOURNAL
Open-Source Accounting with SQL-Ledger
All-Linux Recording Studio
The End of E-mail As We Know It
spam 1994-2004
Moving 35,000 users to Postfix and OpenLDAP
World's simplest blog: blosxom
Everything you ever wanted to monitor Superkaramba

Archive Index

Advanced search

Transactions and Rollback with RPM

James Olin Oden

Issue #121, May 2004

When an upgrade works, you get a few more features or better performance. When an upgrade fails, you're in for a weekend of pain. Now, here's how to back off to the old version and keep the system up.

How many times have you installed a great new piece of software only to find you really didn't want to install it after all? To make matters worse, when you installed this software, you had to upgrade several other software packages and install additional ones from scratch. To put things back the way they were, you had to locate earlier versions of the upgraded packages from a multitude of sources, downgrade to these versions and remove any newly installed packages. Of course, if you did not keep a good record of what packages actually were changed and what their previous versions were, things got even worse. Wouldn't it be great if you instead could push one button or run a single command and roll back this upgrade?

In some environments the ability to roll back an upgrade quickly is not only desirable, it is a requirement. For instance, when upgrading a telecommunications company's equipment, software and hardware vendors are required to upgrade equipment in a limited time frame, known as a maintenance window. In this same maintenance window, they also must be able to back out any changes made by the upgrade. Failure to back out an upgrade within the maintenance window results in strict financial penalties.

Rolling Back an Upgrade the Hard Way

As desirable as an automated rollback of RPMs is, RPM has not supported this option until recently. To be fair, RPM has supported downgrading a set of packages. For instance, if you upgraded some RPM *foo-1-1* to the version *foo-1-2*, you could use the --oldpackage switch with the **rpm** command to downgrade to a previous version; like this:

```
# rpm -Uvh --oldpackage foo-1-1.i386.rpm
Preparing...                ################# [100%]
Upgrading...
   1:foo                    ################# [100%]
```

If the upgrade to *foo-1-2* did not require you to upgrade or install any additional RPMs, the --oldpackage switch worked fine. All you had to do was find the original *foo-1-1* RPM, and you were home free. If, on the other hand, you did need to install or upgrade other RPMs on which *foo-1-2* depended, you then had to search for those RPMs in various locations—on your install media, on your distribution's errata site, in various RPM repositories or on various project Web sites.

Once you had hunted down all the dependent RPMs, you would need to downgrade all the ones you had upgraded and erase the fresh ones you had installed. If, instead, you reversed this order and erased the fresh RPMs you had installed before you downgraded the RPMs that had been upgraded, you were greeted with errors from RPM complaining that these packages were required by *foo-1-2*. In short, the old way of rolling back a set of RPMs was painful and fraught with error.

Transactional Rollbacks to the Rescue

Early in 2002, Jeff Johnson, the current maintainer of RPM, began to remedy this rollback problem when he included the transactional rollback feature into the 4.0.3 release of RPM. This feature brought with it the promise of an automated downgrade of a set of RPMs. Like many new features, it was rough around the edges and completely undocumented, except for a few e-mails on the RPM mailing list (rpm-list@redhat.com). Over the past year and a half, transactional rollbacks have matured steadily. In the current RPM 4.2 release, which comes with Red Hat 9, transactional rollbacks are quite usable.

How RPM Transactional Rollbacks Work

Under the hood, RPM treats any set of RPMs it installs as a discrete transaction. This is true when installing one RPM by itself (a transaction of one RPM) or several RPMs simultaneously. Each of these transactions is given a unique transaction ID (TID). As each RPM is installed or upgraded, its entry in the RPM database is marked with the TID of the transaction within which it was installed. This allows RPM to track within which transaction each RPM was installed or upgraded.

To roll back an RPM transaction set, RPM must have access to the set of RPMs that were on the system at the time the transaction occurred. It solves this problem by repackaging each RPM before it is erased and storing these repackaged packages in the repackage directory (by default, /var/spool/

repackage). Repackaged packages contain all the files owned by the RPM as they existed on the system at the time of erasure, the header of the old RPM and all the scriptlets that came with the old RPM.

You may be wondering how this design helps with upgrades. After all, if you upgrade an RPM you're not erasing it. You are erasing it, though, because upgrading an RPM has two parts: the new package is installed, and the old package is erased. This means every time you upgrade a set of packages, RPM first repackages all packages being updated, then installs all the new packages and, finally, erases all the old packages. When RPM repackages the old packages, it also marks the repackaged packages with the TID of the running transaction. The end result is you don't have to scour the Net, media or backups for the RPMs you updated. Because the repackaged packages contain the files that were currently on your system at the time of the upgrade, the need to restore configuration files from backup is eliminated. As a side effect, the md5 checksums of the files in a repackaged package are likely to be wrong, because RPM does not recalculate each checksum when creating the repackaged package. This is not a problem for RPM when it rolls back transactions, but you need to use the --nodigest option to manipulate repackaged packages directly.

Once the repackage directory is populated, RPM requires only a rollback target (the date to which it is rolling back) to perform the rollback. RPM then determines by TID which transactions have been applied to your system since the rollback target date. Next RPM takes this set of transactions, sorts them in the order of most recent to least recent and does the following for each one:

- Finds all the repackaged packages that are marked with this TID.
- Finds all the currently installed packages that are marked with this TID but do not have corresponding repackaged packages.
- Builds a rollback transaction. Repackaged packages are added to this transaction as install elements, and installed packages that have no corresponding repackaged package are added as erase elements.
- Runs the newly built rollback transaction.

By repeating these actions for each transaction from the most recent one to the one nearest or equal to the target date, RPM walks through all transactions that have occurred since the rollback goal and undoes them.

How to Use RPM Transactional Rollbacks

You may be thinking this process is complicated, but using transactional rollbacks actually is rather easy. As a simple example, let's install a single RPM and roll it back. The most crucial point you have to remember is that whenever you do an upgrade or simple erase, you must tell RPM to repackage the old package before it is erased. To do this, use the --repackage option:

```
# rpm -Uvh --repackage foo-1-2.noarch.rpm
Preparing...    ############################## [100%]
Repackaging...
   1:foo         ############################## [100%]
Upgrading...
   1:foo         ############################## [100%]
```

Using this option, RPM first repackaged the old package and then upgraded the new one. On an erase, you also need to use the --repackage option, like this:

```
# rpm -e --repackage foo
```

RPM does not show any output from an erase, but if you look in the repackage directory after an erase, a repackaged package is there.

To roll back this RPM transaction, use the --rollback option followed by the rollback target. The rollback target can be an actual date or something like one hour ago (the date specifier allows the same date formats as the cvs(1) command's -D option). So, if an hour after upgrading foo, you decide you don't want it, you could type:

```
# rpm -Uvh --rollback '2 hours ago'
Rollback packages (+1/-1) to
Thu Jul 31 23:26:52 2003 (0x3f29ddfc):
Preparing...   ########################### 100%]
   1:foo        ########################### [ 33%]
```

The output `Rollback packages (+1/-1)` shows that RPM is going to add one package, the previous version, and erase one package, the currently installed version.

How to Set Up a System for Rollbacks

Transactional rollbacks are only as good as your local repackaged packages repository. One quick way of making them fail is to upgrade or erase something without using the --repackage option. From my experience, it is pretty easy to forget to use this option. Therefore, if you are going to use transactional rollbacks, you want to configure RPM to repackage all erasures automatically. Do this by setting the `%_repackage_all_erasures` macro to 1 in your /etc/rpm/macros file. If the file does not exist simply create it:

```
%_repackage_all_erasures 1
```

By default, RPM does not roll back a newly installed package; that is, it does not erase packages that were not on the system at the time of your upgrade. You probably don't want this to be the default behavior, so you need to tell RPM to allow for this. To do this, set the macro %_unsafe_rollbacks to the date beyond which you do not want to allow an RPM to be completely erased on a rollback. A good value for this is some time after your system's initial install. This date should be in seconds since epoch. To convert a date to seconds since epoch, use the date command:

```
date --date="8/1/2003" +%s
1059710400
```

If you wanted to tell RPM not to remove packages completely that were installed on or before 8/1/2003 (the date in the above example), you would add the following to the /etc/rpm/macros file:

```
%_unsafe_rollbacks 1059710400
```

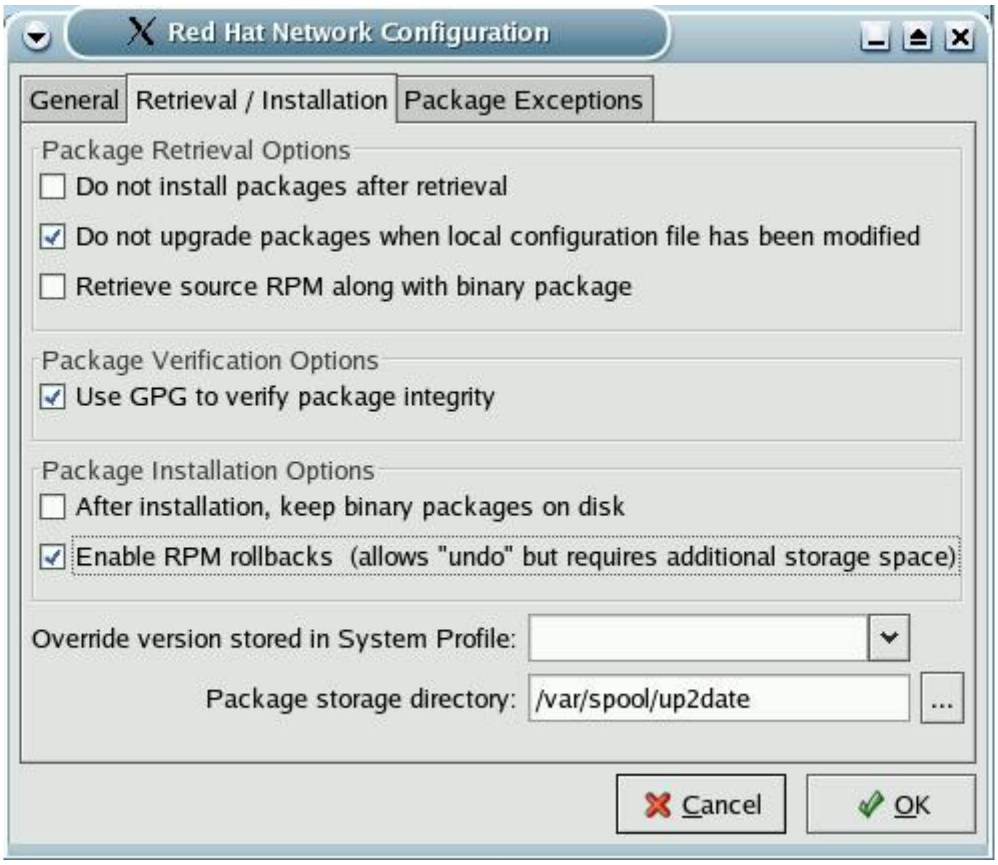
The only other thing you may want to configure is where RPM puts the repackaged packages. One reason for doing this is to ensure they are placed in a partition that has ample space. To change the repackage directory, set the %_repackage_dir macro to the directory where you wish to store the repackaged packages:

```
%_repackage_dir /my_rp_repository
```

Now you have a system that automatically repackages all erasures (so you or someone else does not forget), erases newly installed packages on a rollback (but won't erase your whole system) and places the repackaged packages where you want to store them.

Using up2date to Roll Back RPM Transactions

In Red Hat 9, up2date supports rollbacks using RPM's transactional rollback mechanism. Configuring it to support transactional rollbacks is as simple as running `up2date-config`, clicking the Retrieval/Installation tab and then clicking the Enable RPM rollbacks check box (see Figure 1). You have to configure RPM itself as described in the previous section. When you upgrade your system using up2date, once you have configured both RPM and up2date, RPM creates the repackaged packages of RPMs you are updating before it upgrades those packages.


Red Hat Network Configuration

General | Retrieval / Installation | Package Exceptions

Package Retrieval Options
- [ ] Do not install packages after retrieval
- [✓] Do not upgrade packages when local configuration file has been modified
- [ ] Retrieve source RPM along with binary package

Package Verification Options
- [✓] Use GPG to verify package integrity

Package Installation Options
- [ ] After installation, keep binary packages on disk
- [✓] Enable RPM rollbacks (allows "undo" but requires additional storage space)

Override version stored in System Profile:

Package storage directory: /var/spool/up2date

Figure 1. Enabling RPM Rollbacks in up2date

To list the different known rollback targets, type:

```
up2date --list-rollbacks
```

You should receive a listing like this:

```
# up2date --list-rollbacks
install time: Sun Jul 27 20:49:55 2003   tid:1059353395
                [-] goo-1.0-1.0:

install time: Tue Jul 29 20:44:25 2003   tid:1059525865
                [-] foo-1-2:
```

This command is handy even if you are not actually using up2date, because the rpm command does not provide a way of displaying such information.

To undo a transaction, use the --undo option, which undoes the last transaction that was installed. Simply type:

```
# up2date --undo
```

If you want to roll back multiple transactions, run this command multiple times. The ability to roll back from the GUI is not supported.

Auto-Rollback Patch

RPM normally delivers packages using a best effort strategy, meaning if one or more RPMs fails to install, the remaining RPMs in the transaction still are installed. This is a desirable behavior in some environments, but in others it would be much better if, instead, RPM automatically rolled back the failed transaction. Because I work in such an environment (telecommunications), I wrote a patch called the auto-rollback patch. This patch allows you to configure RPM such that if a transaction fails, RPM automatically rolls back the failed transaction. It does leave behind the failed RPM if it failed in its %post scriptlet; hopefully that soon will be fixed (patches anyone?).

If you would like to use this feature, you can download the patch (or RPMs that have the patch applied) from lee.k12.nc.us/~joden/misc/patches/rpm. Once you have a version of RPM installed with the auto-rollback patch, you need to configure RPM to use the auto-rollback feature. To do so, edit /etc/rpm/macros and add the following macro definition:

```
%_rollback_transaction_on_failure 1
```

After doing this, the next time you install/upgrade a set of RPMs and one fails to install, RPM automatically rolls back the failed transaction, except for the failing one if it failed in the %post scriptlet.

Conclusion

RPM transactional rollbacks provide an efficient way of undoing RPM upgrades. They also provide a solid building block upon which system update programs (such as up2date, yum and apt-get) can provide automated rollback functionality. However, transactional rollbacks are not for everyone. To quote Jeff Johnson, “the --rollback option...requires absolutely perfect system administration and is mostly mechanism, not policy.” Transactional rollbacks are an all-or-nothing affair. Care must be taken to ensure that all erasures are repackaged, as RPM's ability to roll back transactions is only as good as its source of repackaged packages. The administrator must ensure that extra space is allocated for the storage of the repackaged packages. Finally, RPM's transactional rollback feature is a work-in-progress. That said, RPM transactional rollbacks have come a long way from their beginnings. If you want to ensure that RPM updates to a system can be undone quickly, they may be exactly what the doctor ordered.

James Olin Oden (joden@lee.k12.nc.us) is a software engineer at Tekelec. He has administered UNIX-type systems and developed under them for over a decade. He also is the creator of Tech Tracker (tt.lee.k12.nc.us), a Web-based IT-tracking system.

Archive Index Issue Table of Contents

Advanced search

HEC Montréal: Deployment of a Large-Scale Mail Installation

Ludovic Marcotte

Issue #121, May 2004

Forced to upgrade by a flood of junk mail, this university went to a heavy-duty system based on Linux. And, they made it a seamless transition without disturbing users' existing mail configurations.

Over the past few years, e-mail has grown into one of the most important communication mediums. Naturally, e-mail infrastructures must be fast, secure and reliable. Ideally, they also should be able to integrate easily and effectively with anti-unsolicited bulk e-mail (UBE) solutions.

HEC Montréal is Canada's first management school, founded in 1907. More than 11,000 students and 220 professors use HEC's e-mail system every year, and alumni keep their e-mail accounts after graduation. Unfortunately, the proprietary e-mail system did not evolve and as the load started to increase, the infrastructure could no longer keep up with requirements.

The previous mail infrastructure at HEC Montréal was based on four IBM AIX servers running Netscape Messaging Server 4.15. Each of those servers offered all services (IMAP, POP3, SMTP and Webmail access) for a subset of users. The system simply did not scale to current mail requirements. According to Eddy Béliveau, Senior Network Analyst at HEC Montréal:

> We found ourselves with mail server software that had not been upgraded in the last two years because the AIX platform was no longer supported by Sun/iPlanet/Netscape, which owned the mail server software. We had a regular increase of our e-mail traffic during the last 12 months due to the presence of UBE and viruses trying to replicate themselves. We got peaks of over 100 concurrent SMTP connections, which was too much for our servers; the typical load average was over 50 on all servers. We could not, on our old

> 133MHz servers, execute any anti-virus or anti-UBE applications, not even a simple RBL filtering policy. Thus, we had to re-examine the hardware and software architecture of our e-mail system but [could] not find time to install alternatives. We were like a dog running after his tail trying to stabilize the situation.

HEC Montréal contacted us at Inverse, Inc., to help them replace the mail infrastructure and deploy a better alternative.

Figure 1. HEC Montréal is a tough e-mail problem: 35,500 users and more than 600,000 spam messages a week.

The Proposed Solution

The proposed solution was driven by the following factors:

- Cost: HEC Montréal could not afford a per-user license fee for 35,500 users.
- Ease of maintenance: the infrastructure had to be easy to manage. Accounts creation and destruction should be automated, updates should be easy to apply and the infrastructure should let HEC Montréal leverage the expertise they have.
- Security: the components of the solution should have a proven security track record.
- Robustness: the components should be mature and should have been used in production environments for months. Furthermore, the

development should be active to accelerate bug fixes, feature enhancements and security updates.

- Scalability: the solution must meet its purpose for many months, because the number of users grows by 2,000–3,000 every year. Its architecture also should allow adding extra servers to distribute the load or offer more redundancy.

When we were first approached, HEC Montréal was leaning toward a Linux-based solution running Novell NetMail 3.1. Having great experience with free alternatives, we decided to compare the solution we had in mind with Novell's offerings.

That said, we built two identical test environments using Red Hat Linux 9 and installed NetMail 3.1 on one and our proposed solution on the other. Next, we performed a series of stress tests in order to measure the stability and the performance of the two solutions. The tests were performed with two benchmarking utilities, postal and tm. The results showed that while NetMail was the fastest for POP3 operations, it proved to be the slowest in the IMAP and SMTP tests. It also had a lot of stability issues when overloading the server with IMAP requests.

Combined with our experience, we proposed a solution based almost entirely on open-source components. We started with a standard Red Hat Linux 9 distribution using Silicon Graphics, Inc.'s XFS kernel packages. We included Cyrus IMAP and Cyrus SASL, which included IMAP, LMTP and POP3 dæmons as well as authentication libraries and redirection/vacation scripts support using Sieve. Next, Postfix, AMaViS, SpamAssassin, Vipul's Razor and NAI VirusScan were added to build a complete SMTP server solution with enhanced tools to limit the delivery of UBE and viruses. Apache, PHP4, IMAP Proxy and SquirrelMail provided a complete Webmail solution. OpenLDAP was added to store all information regarding users' accounts (e-mail address and aliases, SquirrelMail preferences and so on), as well as other specific attributes of HEC Montréal. Finally, we installed Linux HA Heartbeat, software used to monitor the health of some nodes on the network.

The new infrastructure is running on 11 IBM eServer xSeries x305 and x335 servers. The two x335s are connected to an IBM FAST 700 Storage Array Network (SAN) using Fibre Channel, where the mailstore resides. The XFS filesystem is used for the mailstore in order to maximize file access operations. Figure 2 depicts the architecture.

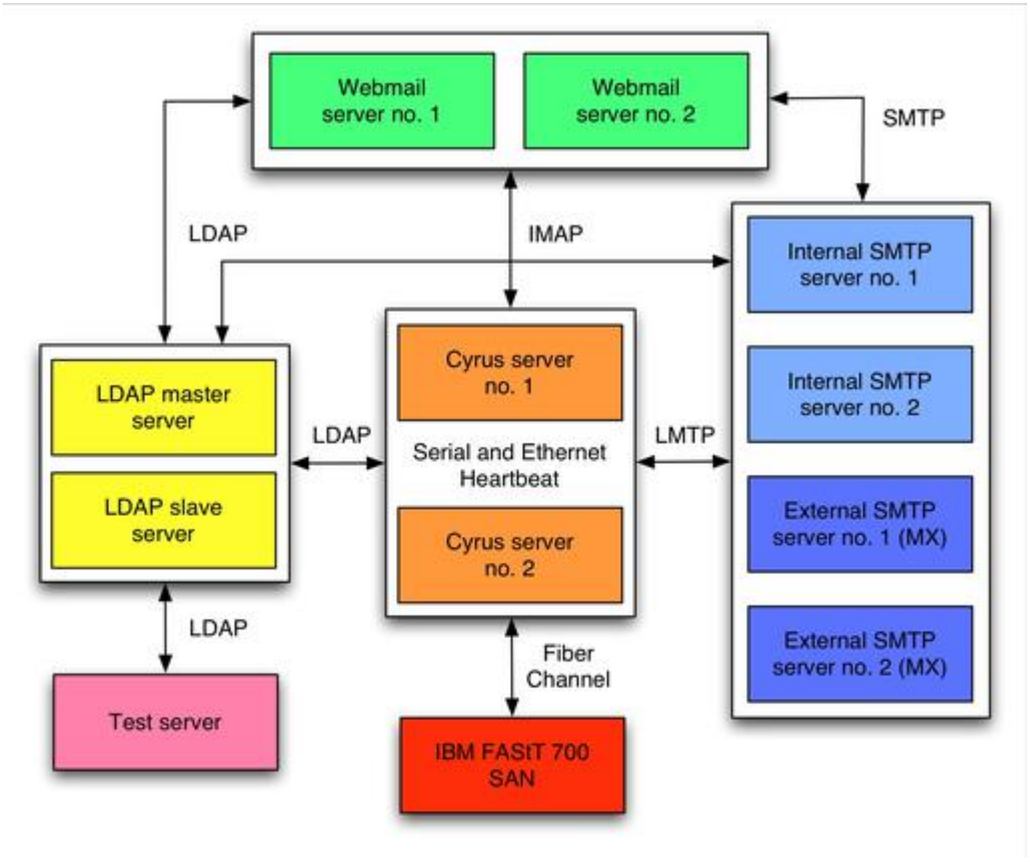


Figure 2. Architecture of the Proposed Infrastructure

Four STMP servers running Postfix are used: two of them are mail exchangers (MXes) for the HEC Montréal domains and the other two serve internal mailing needs. These servers also use AMaViS, SpamAssassin, Vipul's Razor and Network Associates' VirusScan to limit the delivery of UBE and viruses. Furthermore, two Cyrus IMAP servers are connected using serial and Ethernet cables for high availability. Only one Cyrus IMAP server is active at any moment; it serves all POP3 and IMAP connections, stores mails on the SAN (received using the LMTP protocol from the four Postfix servers) and processes Sieve scripts.

Two Webmail servers run Apache, PHP4, SquirrelMail and IMAP Proxy. The latter is used to cache IMAP connections between SquirrelMail and the Cyrus IMAP server in order to minimize the load (authentication and process forks) on the mailstore. Finally, one other server is used only for testing purposes. That is, any modifications to the infrastructure must go through this server, which is configured to run every component, before being applied to the environment in production.

With regard to the UBE filtering, we check mail at many levels to ensure we block as many as we can. Our checks include carefully chosen real-time blackhole lists (RBLs); header and MIME header checks using up-to-date maps from SecuritySage, Inc.; and content filtering initiated from AMaViS using SpamAssassin, Vipul's Razor for UBEs analysis and VirusScan for viruses.

This solution has proven to be greatly effective and produces few false positives. The system also was built with load balancing and failover in mind. The SMTP and the Webmail servers are used in a round-robin fashion, efficiently distributing the load among all of them.

The main Cyrus server has an identical backup server in case of failure. The latter is connected to the main Cyrus server and uses Heartbeat to monitor the availability of the server. In case of a failure (hardware problem, operating system crash and so on), the secondary Cyrus server takes over all services. Heartbeat automatically mounts the mailstore (located on the SAN), activates the network alias and starts all Cyrus services. This offers a warm switch-over that minimizes the outage time; sometimes it's not even noticeable.

Finally, the LDAP system offers a master node together with a slave that replicates the former using slurpd. All services are configured to failover automatically to the slave in case of a failure on the master node. Some services also are configured to use the slave as the master node in order to distribute the LDAP load among both servers; they failover to the master node.

Migration

After putting the 11 servers for the new infrastructure in place, one of the remaining challenges was to migrate all users from the old infrastructure to the new one. About 35,500 users, 82,500 mailboxes and hundreds of thousands of messages (35GB of mail) had to be migrated. Furthermore, redirection scripts and vacation messages also had to be converted, and information such as preferences from the previous Webmail system had to be kept intact. In order to do this, we created a set of Perl scripts to take care of the entire migration in a way that would appear seamless for the users:

- LDAP Init: populates the new LDAP server (based on OpenLDAP) using the values from the previous LDAP server (based on Netscape iPlanet). Included attributes are e-mail addresses and aliases, special folders and signature preferences for Webmail.
- Create Users: creates all user accounts about to be migrated.
- Load Sieve: creates Sieve scripts and uploads them to the mailstore by reading attributes from the previous LDAP server. Sieve scripts are used for automatic redirections and vacation messages.
- Copy Mailboxes: copies all mailboxes for the users being migrated. All message flags are kept intact. The IMAP protocol is used a lot in this script. This script also updates the mailHost attribute on both LDAP servers so the mails are routed to the correct destination mailboxes.
- Update Mailboxes: run the morning after the migration to move the remaining (if any) messages in the users' mailboxes. Mail could have been

stuck in the queue of the SMTP servers, before the users' mailHost attributes were changed.

To minimize service interruptions for the users, we ran the scripts in the order listed once classes were finished at the end of the day. Few messages were rejected during the import process; those that were simply were retried by the source SMTP servers. In total, four nights were required to migrate all the information. Running the scripts took from four to seven hours, depending on the number of users located on each source server and the execution speed, which was mainly limited by the performance of the old AIX servers.

Key Statistics

After the migration, we extensively monitored all services in order to discover any problems. As expected, we didn't have many. We mainly tuned the minimum preforks of Cyrus processes as well as their respective maximum children. We also tuned the SMTP servers for the default process limits and preforks for AMaViS. We also used temporary LDAP queries during the migration, so we had to replace them with optimized ones once the migration finished.

During a typical day, HEC Montréal receives over 125,000 e-mails, and 60% to 80% of the traffic is composed of UBEs. The internal SMTP servers also manage thousands of messages sent by users, distribution lists or other systems. About 300,000 POP3 connections (from 5,500 different users) and 60,000 IMAP connections (from 5,000 different users) are initiated every day on the main Cyrus server. Peaks of 225 concurrent IMAP connections and 50 concurrent POP3 connections frequently are encountered.

As mentioned earlier, the anti-UBE policies in place have proven to be effective. During the first week after the migration, the two mail exchangers blocked more than 600,000 unsolicited bulk e-mails. The week after, spammers were less aggressive and the systems blocked over a quarter of a million messages. The most effective policy is the RBL checks, followed by the content filtering checks (using SpamAssassin and Vipul's Razor) and, finally, the header and MIME header checks.

To extract those statistics, we installed Spamity, which parses mail logs from the four Postfix servers and updates a PostgreSQL database running on the test server. Thereafter, users or administrators can examine the mail that was blocked by anti-UBE policies by using a simple Web browser. Users also can perform searches for specific e-mail addresses or domain names and filter the results by anti-UBE policies.

Conclusion

As you have seen in this article, migrating from a proprietary solution to an open-source solution was a challenge. According to Emmanuel Vigne, Information Systems Director at HEC Montréal:

> The key business benefits are huge, as we nearly eliminated UBE and greatly enhanced the architecture of our mail infrastructure. We moved from an architecture where all services were offered by four servers to an architecture where the services are offered by many servers. That allows us to minimize any potential outage and scale as the number of users grow. In case of a failure, only one specific service is affected, contrary to the situation before where thousands of users could no longer use the e-mail service in case of a single server failure.

Putting this new infrastructure in place allowed us to contribute to the Open Source community by developing a set of patches to correct bugs and/or add features to most components we installed.

As with any other system, this one will evolve over time. Interesting anti-UBE technologies are emerging, such as Sender Policy Framework (SPF) [see page 50] and Spamhaus Exploits Block List (XBL), and a new stable version of Cyrus is available with NNTP and mailbox annotations support. In addition, Postfix 2.1 is coming along nicely and should offer excellent connection/rate control with its new anvil server.

Finally, as this article was being written, a mirroring solution was being deployed for the SAN. This should offer storage redundancy and eliminate the single potential point of failure in the current infrastructure.

Resources for this article: /article/7456.

Ludovic Marcotte (ludovic@inverse.ca) holds a Bachelor's degree in Computer Science from the University of Montréal. He currently is a software architect for Inverse, Inc., an IT consulting company located in downtown Montréal.

Advanced search

SPF, MTAs and SRS

Meng Weng Wong

Issue #121, May 2004

Last month, we learned how to flag outgoing e-mail as authentic using DNS. Now, it's time to check incoming mail and protect our users from forged spam and worms.

Sender Policy Framework (SPF) takes aim at the practice of return-path spoofing, a technique employed by worms, viruses and other senders of unwanted mail. SPF consists of two parts. First, domain administrators publish SPF records in the DNS. Those records describe the servers the domain uses for outbound mail. Then they are read by SPF-enabled MTAs. Mail coming from a server not described in SPF can be considered forged.

This article, the second of a two-part series, explains how to add SPF capabilities to your mail server. It also discusses how e-mail forwarding and Web-generated e-mail services can adjust to SPF by performing sender rewriting.

This article was written in early February 2004 and reflects the state of the Internet at that time. The MyDoom worm, a virus that spoofs return-path addresses, recently had littered millions of mailboxes with bogus bounce messages.

It's Your Turn

Last month I described how to construct an SPF record, and DNS administrators all over the world responded. First, they published records, then crossed their fingers and waited. What are they waiting for? They're waiting for you. They've made it possible for you to distinguish their legitimate mail easily from forgeries. Now it's your turn to help them cut down on bogus bounces and abuse reports. If you're just tuning in, see the on-line Resources section for last month's article and an easy Web-based SPF wizard.

Adding SPF to Your MTA

The major mail transfer agents (MTAs) in the Linux world are Sendmail, Postfix, Qmail and Exim. Although most antispam vendors already have SPF support included in their products or plan to add it in their next release, MTAs tend to want to leave that task up to you. Most MTAs offer an interface into which you can plug your antispam tool.

SPF can be made to work in your MTA in two ways. If you're the kind of sysadmin who prefers to compile your own software, start at the SPF downloads page. There you can find the SPF plugin that's right for your MTA, plus detailed installation instructions. If you prefer to manage your software using a package system, you may find an SPF-enabled version of your MTA already built and ready to install.

Most of the plugins rely on the reference Perl library Mail::SPF::Query. You can install that library directly from CPAN, or you can try to find a package for it. It provides a simple program to run SPF queries at the command line. It also provides a simple dæmon that handles SPF query requests over a UNIX domain or inet socket.

By default, most of the plugins tell the MTA to reject messages that fail SPF tests and add a Received-SPF header to the rest. Conservative installations may prefer to add the line `Received-SPF: fail` instead of rejecting. This configuration option is described in the plugin documentation.

Sendmail

Sendmail's plugin interface is called Milter (see on-line Resources). Recent Sendmail versions have Milter capability compiled in by default. Sendmail talks to Milter through a socket interface. Sendmail tells Milter about the incoming SMTP transaction, and Milter tells Sendmail what to do. Milter runs as a dæmon and needs to be started separately.

Two Milters should be available at the SPF Web site: one in Perl and one in C. The Perl version is a little more mature, but if you need speed, the C version may be a better choice.

To make Milter work with Sendmail, add a couple of lines to your sendmail.mc file, rebuild sendmail.cf and restart Sendmail.

If you'd rather not use Milter, libspf comes with a patch that integrates SPF directly into Sendmail.

Postfix

Postfix 2.1 comes with a policy dæmon interface. It works much like Milter does: Postfix connects to the dæmon and provides a play-by-play commentary, and the dæmon returns an action to Postfix. If you're running a recent development snapshot of version 2.0, make sure you're using 2.0.18-20040122 or later.

Policy dæmons are configured in main.cf and master.cf. They are managed by Postfix, which starts and stops them as needed, so you don't need to worry about that. The Postfix policy dæmon is written in Perl and calls the standard Mail::SPF::Query library.

Exim

Exim 4 introduced Access Control Lists (ACLs), a powerful and compact mini-language for making antispam and other local policy decisions. The ACL code that handles SPF for Exim is only about 12 lines long.

You need to install the Mail::SPF::Query library and run its SPF dæmon, which listens on a socket. The SPF ACL connects to the spfd and reads it the client IP, HELO argument and MAIL FROM sender address. It then receives an SPF result, a response for the SMTP server and a Received-SPF header line. You need to start the spfd separately.

Qmail

Qmail does not have the same kind of plugin interface that the other MTAs do. Instead, SPF provides a patch that integrates SPF directly into Qmail. In addition, many Qmail users screen their mail with qpsmtpd: if you do, SPF is a plugin you can turn on easily.

James Couzens is the primary author of the C SPF library. libspf comes with a patch for Qmail and for other MTAs as well.

Testing the Plugin

Once you've installed the plugin and turned it on, you should perform two tests. First and most important, legitimate mail needs to get through. If something broke, maybe you're not running something you need to—double-check. If it's still broken, back out the patch and report your experience to the spf-help mailing list.

Second, confirm that forged mail is rejected. If you can speak SMTP by hand, engineer a message with MAIL FROM:<linuxjournal-test@altavista.com>. The domain altavista.com is not used for mail, so it always returns a FAIL message.

They have asked that test messages contain the word test. This can be tricky to execute because if they recognize a trusted client, both your MTA and SPF will turn a fail into a pass. Therefore, don't telnet to localhost; use your machine's actual hostname, and if possible try to open the connection from an outside host. If you receive a 550 response and an error message that refers to spf.pobox.com/why.html, it's working.

If you use a secondary MX, tell your SPF client not to reject its mail. How to do this is described in detail in the installation instructions for your plugin.

Received-SPF: What the Codes Mean

You should notice that your mail now contains a Received-SPF header that carries a number of result codes:

- NONE: the domain does not publish SPF records. Your MTA should proceed as usual.
- PASS: the mail is not forged, but that doesn't mean it's legitimate. Remember, spammers can publish SPF too. You still should test its domain against a right-hand-side block list (RHSBL). But if the sender is on your trusted whitelist, you can skip further antispam checks with confidence.
- FAIL: the mail is a forgery, and you can reject it with confidence. There is a miniscule chance the message is legitimate but was sent by a misconfigured sender. In that case, the error message they receive tells them they need to configure their MUA with SMTP AUTH. SPF's design philosophy is that it's better to fail obviously with a hearty error message than to risk silently burying mail in a spambox.
- SOFTFAIL: the message could be a forgery, but the domain's ISP is working on switching its users to SMTP AUTH, so the message could be legitimate. You should accept the message, but subject it to more stringent antispam checks.
- NEUTRAL: the domain just has started down the road to SPF, and their default response is `?all`. They would like you to pretend the response was `NONE` while they consider moving the default toward `SOFTFAIL` and `FAIL`. Big ISPs with millions of users move slowly; it's not their fault.
- ERROR: there was a temporary DNS lookup error. Normally, your MTA should return a `450 temporary failure` when this happens.
- UNKNOWN: a permanent error caused the SPF lookup to abort; perhaps there was a syntax error in the record, or maybe the record pointed to another domain that doesn't have an SPF record.

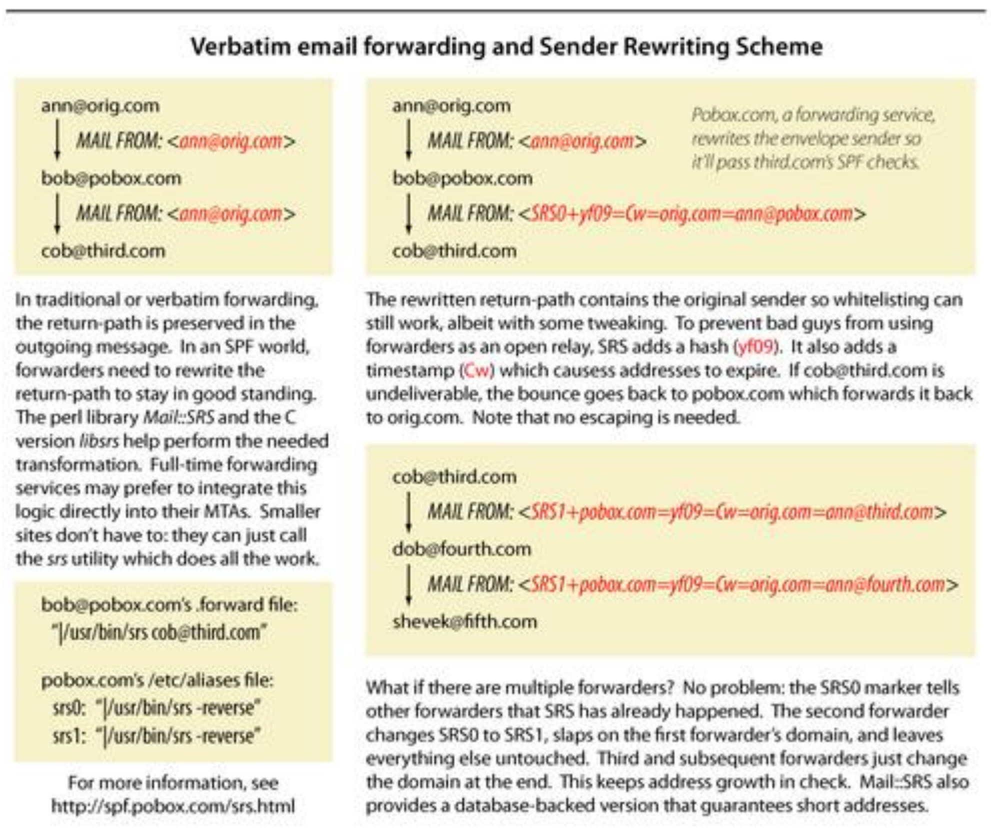
The Price of SPF

In the past ten years we have grown tremendously dependent on e-mail; we are made aware of just how dependent we are every time a worm hits. Analysts routinely announce that spam and viruses cost the economy billions of dollars. The success of SPF shows that people are desperate for change.

But, change has its own price. If there were such a thing as a painless solution to spam, we already would have adopted it. The war on spam has dragged on so long in part because the best experts on spam simply could not agree on exactly what trade-offs they wanted to make, but that phase of debate is drawing to a close. In every antispam future they have discussed, sender authentication is the first and fundamental step. Now, many possible sender authentication models are available, but the designated-sender scheme that SPF provides is probably the easiest to implement.

Cryptography definitely is in our future, but it's not here yet. Like first aid, SPF offers immediate benefit, and it's something we can do right away.

What is the price of SPF? Every designated sender scheme breaks two things. First, SPF breaks verbatim e-mail forwarding (Figure 1). Services that provide permanent e-mail addresses, such as pobox.com, are used to forward mail the way UNIX .forward and /etc/aliases files do. When the mail leaves their servers, the return-path address in the envelope is unchanged. But in an SPF world, resent messages now look a lot like forgeries. To fix this, forwarding services need to rewrite their return paths. So do other sites that depend on .forward and /etc/aliases to send mail off-site.


Verbatim email forwarding and Sender Rewriting Scheme
ann@orig.com
MAIL FROM: <ann@orig.com>
bob@pobox.com
MAIL FROM: <ann@orig.com>
cob@third.com
ann@orig.com
MAIL FROM: <ann@orig.com>
bob@pobox.com
MAIL FROM: <SRS0+yf09=Cw=orig.com=ann@pobox.com>
cob@third.com
Pobox.com, a forwarding service, rewrites the envelope sender so it'll pass third.com's SPF checks.
cob@third.com
MAIL FROM: <SRS1+pobox.com=yf09=Cw=orig.com=ann@third.com>
dob@fourth.com
MAIL FROM: <SRS1+pobox.com=yf09=Cw=orig.com=ann@fourth.com>
shevek@fifth.com

In traditional or verbatim forwarding, the return-path is preserved in the outgoing message. In an SPF world, forwarders need to rewrite the return-path to stay in good standing. The perl library *Mail::SRS* and the C version *libsrs* help perform the needed transformation. Full-time forwarding services may prefer to integrate this logic directly into their MTAs. Smaller sites don't have to: they can just call the *srs* utility which does all the work.

```
bob@pobox.com's .forward file:
  "|/usr/bin/srs cob@third.com"

pobox.com's /etc/aliases file:
  srs0:  "|/usr/bin/srs -reverse"
  srs1:  "|/usr/bin/srs -reverse"
```

For more information, see http://spf.pobox.com/srs.html

The rewritten return-path contains the original sender so whitelisting can still work, albeit with some tweaking. To prevent bad guys from using forwarders as an open relay, SRS adds a hash (yf09). It also adds a timestamp (Cw) which causess addresses to expire. If cob@third.com is undeliverable, the bounce goes back to pobox.com which forwards it back to orig.com. Note that no escaping is needed.

What if there are multiple forwarders? No problem: the SRS0 marker tells other forwarders that SRS has already happened. The second forwarder changes SRS0 to SRS1, slaps on the first forwarder's domain, and leaves everything else untouched. Third and subsequent forwarders just change the domain at the end. This keeps address growth in check. Mail::SRS also provides a database-backed version that guarantees short addresses.

Figure 1. Old-school e-mail forwarding breaks under SPF.

The solution is called SRS, sender rewriting scheme. It encapsulates the original sender address in the rewritten, SPF-compliant, return address. If a message should bounce, it comes back to you, and you unwrap the address and forward the bounce back to the sender. Forwarding services would have to do this even in a world without SPF, because ISPs already are performing pseudo-SPF checks. SPF simply gave everyone a standard way to do what they already were doing piecemeal. In the same way that responsible sites closed down their open relays over the past few years, in the coming months responsible sites will begin to operate SRS-compliant forwarding; pobox.com already is doing SRS, and other forwarding services are expected to follow.

The good news is the community that developed SPF already has produced SRS code for your MTA. Those patches are available from the same place you got your SPF patches. By the time you read this, they even might be bundled into your MTA. The goal is for the average installation to be able to upgrade to the latest version and have SRS magically work (see Resources).

So, this solves the e-mail forwarding problem. Getting SRS into the field is simply a matter of time. But SPF also breaks Web-generated e-mail. Greeting card sites and "e-mail me this news article" sites tend to use your e-mail address not only in the From: header but in the envelope sender too. In SPF terms, that kind of behaviour is indistinguishable from forgery.

To solve this problem, those sites can do one of two things. First, if the mail they send isn't that important, they can set the return-path address to nobody@example.com and eat the bounces. Newer, more progressive sites, such as Orkut, already do something like this. But if the mail is important, was sent on behalf of a user who was logged in to the Web site properly, and if the Web site had previously confirmed the user's e-mail address, then the Web site could perform SRS *on itself*—encapsulating the user's return address so that bounces would be properly forwarded.

What about the transition period, you ask. Won't there be a time of disruption while the forwarders groan their way toward SRS-compliance? What about the sites that are unwilling or slow to adapt?

Well, here's a little secret. We have a fairly good idea who the major culprits are; we know, for instance, that eBay sometimes sends mail with a legitimately forged envelope return-path. The people who developed SPF use eBay, too, and they don't want to lose e-mail any more than you do. So they came up with a hack. They set up a whitelist that identifies all these legitimate forgers; pobox.com is on the list, as are acm.org, eBay and the newspaper Web sites that do "e-mail me this article".

Every SPF client we've talked about in this article knows about that whitelist. Every SPF client we know of gives that whitelist a chance to override a fail. If your mother sends mail from her AOL account to your acm.org address, your SPF client accepts that message, even though it's technically a forgery. (If you get forwarded mail through a system that's not on the list—from, say, a friend's home Linux box—you should whitelist that box in your MTA.) When acm.org implements SRS, the problem will go away.

SPF's critics tend to say "it breaks forwarding". The SPF community rose to the occasion and did their best to ease the transition. They offered two solutions, one short-term and one long-term, that meet in the middle. Together they sugarcoat the bitter pill.

Change means pain. The transition to an SPF world won't be painless, but it's like the pain of an injection that makes the illness go away. E-mail is very sick. Some say it will not survive spam, but I don't agree. I think SPF will set it firmly on the road to recovery.

Resources for this article: /article/7465.

Meng Weng Wong is founder and CTO of pobox.com, the e-mail forwarding company, which celebrates its tenth anniversary this year. He is working on a science-fiction novel set on a planet where traditional fantasy magic turns out to be implemented, following Clarke's famous dictum, using nanotechnology.

Advanced search

Policy Routing for Fun and Profit

David Mandelstam

Nenad Corbic

Issue #121, May 2004

Get the bandwidth you need without a surprise bill at the end of the month.

Sangoma is a manufacturer of PCI-based WAN interface cards. For demonstrations and redundancy, we have two separate high-speed Internet connections: a full T1 carrying Frame Relay, using our PCI S5148 T1/E1 modem, and an ADSL connection carrying PPPoE over ATM, using our PCI S518 ADSL modem. The ADSL line is shared with our fax machine, the only telephone line not connected to our PBX. We use two different ISPs for the services. The ADSL and fax telephone line are provided by Bell Canada's Sympatico service, and the T1 Frame Relay connection is provided by MCI.

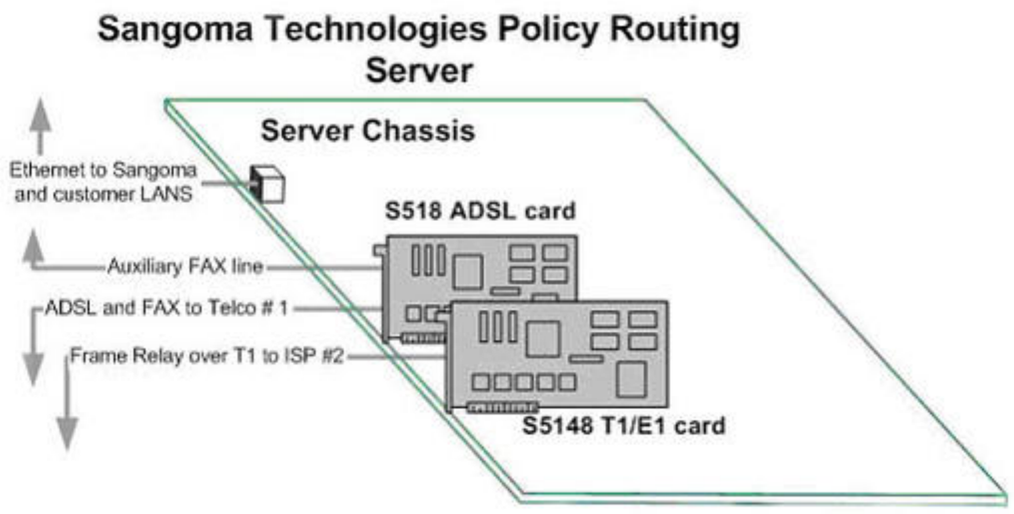

Sangoma Technologies Policy Routing Server
Server Chassis
Ethernet to Sangoma and customer LANS
S518 ADSL card
Auxiliary FAX line
ADSL and FAX to Telco # 1
Frame Relay over T1 to ISP #2
S5148 T1/E1 card

Figure 1. For redundancy and cost control, the policy routing server has both T1 and ADSL connections.

Bandwidth and Costs

The combination of the installed T1 and ADSL Internet links provide reliable connectivity, but the resultant bandwidth is excessive for our requirements.

Sangoma's Web site is hosted by Earthlink in Atlanta, so our Internet access requirements are not too different from any other company's, primarily e-mail and Web access, with some FTP traffic mainly as uploads to the FTP server on our Web site. We could manage without a fixed IP address, although we do find it valuable that the T1 link is provisioned with a range of fixed addresses.

All our Internet servers are Linux-based. Although we support Windows, FreeBSD, Solaris and other popular operating systems, Linux is our most important platform, and only Linux has the rich toolset of traffic management routines we need. The layout is shown in Figure 1.

The ADSL line is inexpensive, especially after the rebate we get for using our own ADSL modem instead of the consumer-grade external ADSL modem normally provided as part of the service. T1 in Canada is expensive as compared to the US; the cost of a standard unrestricted T1 Internet link can exceed $1,900 CAN ($1,450 US) per month.

Sangoma resells Internet access to two other tenants in our building to offset costs somewhat. The other parties sharing our connections host Web and VPN services, so they need both fixed addresses and high outbound bandwidth, which is why they are interested in sharing our T1 line. The private and public segments of the T1 line are shown in Figure 2.

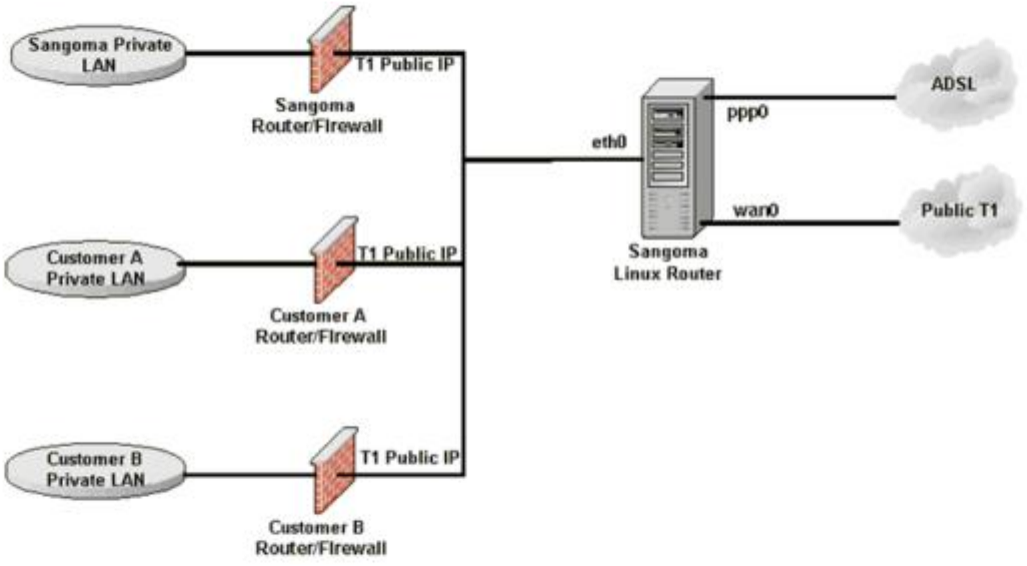


Figure 2. Two tenants in the building buy Internet access from Sangoma.

The two Sangoma Linux machines could be combined into one quite easily. The combination router would have another NIC to support the public network segments to customers A and B. Sangoma's firewall would operate between Sangoma's private LAN segment and all the other public segments, which include the Frame Relay T1 link, the ADSL link and the public Ethernet link.

The financial contributions from customers A and B are not enough to pay for a full T1 at Canadian prices. The solution for us was to employ a usage-based service for T1. This is a so-called burstable T1 service, which is about half the price of a full bandwidth T1 line. The T1 use is unrestricted up to the full bandwidth of 1,536Mbps full duplex. Billing is based on the 95th percentile of the bandwidth used. Traffic is sampled in five-minute intervals, and the average bandwidth for the five minutes is calculated. At the end of the month, these five-minute bins are arranged in decreasing order of bandwidth. The top 5% are discarded, and the billing rate depends on the next highest bin for the month. The trigger throughput in our case is 128kbps. If our 95th percentile throughput exceeds 128kbps, the monthly line charge is incremented by a minimum of $300.

This complicated billing structure is hard for subscribers to understand. It looks like a good deal to the customer but is difficult to manage and hard to measure. The billing rate is measured at the 5% level, where the rate of change of the usage curve is near a maximum. So, many users find themselves paying high bills that depend on the bandwidths of only one or two five-minute bins out of the more than 8,500 bins measured each month. Unless one's traffic is consistently low, one quickly finds that the surcharges are such that one may as well bite the bullet and take the full T1, even though the average throughput may be well below 128kbps.

The major plus is that the highest 5% of bandwidth usage for each month is “free”. This amounts to about 36 hours per month at any bandwidth without penalty, so almost a full working week out of the month is available at full line speed. Figure 3 shows the ideal traffic pattern for achieving the highest throughput for the lowest cost on our burstable T1 service. Essentially, the aim of the traffic control logic is to restrict the T1 bandwidth to 128kbps after the first free 36 hours of unrestricted bandwidth has been consumed in a given month.

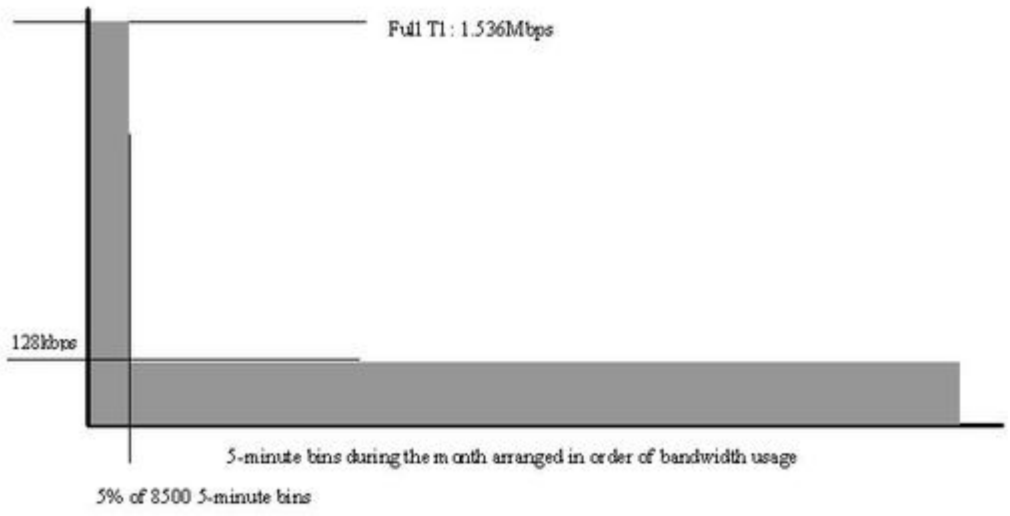


Figure 3. To achieve the lowest possible cost, the ideal T1 traffic pattern uses the full T1 line only 5% of the time.

So how do we manage to take the bait without the hook? With a series of scripts and dæmons that use a combination of policy routing, IP accounting and traffic shaping we can manage the network intelligently, so both we and our co-users get maximum performance at minimum cost.

Policy Routing with iptables and iproute2

The first step is to unload the T1 of all the traffic that could be routed through the ADSL line without losing service quality. Our ADSL line runs at a downstream rate of 1,728kbps, with an upstream rate of 800kbps. The T1 nominal rate is 1,536kbps, full duplex. The ADSL line is less efficient than the Frame Relay T1 line because of the high ATM and error correction overheads. So in terms of useful throughput, the incoming or down rates of the T1 and ADSL lines are similar.

We are fortunate that our particular ADSL connection seems to have a low level of oversubscription, so our performance is more consistent than that of many similar installations. Normally, ADSL links are oversubscribed at the central office end by up to 200 or 300 times, which results in poor performance in peak periods. But even with our near perfect ADSL line, the true upstream rate of the ADSL line is less than half that of the T1. It therefore makes sense to use ADSL for downstream traffic and reserve the T1 for the upstream flow.

Apart from the speed differences, the other major difference between our Frame Relay T1 line and the ADSL line is that the T1 offers a small range of fixed IP addresses, whereas the ADSL line is assigned an IP address by a DHCP server. At a minimum, services that need to support unsolicited incoming traffic on a fixed IP address, such as Web servers, need to be on the T1 line.

Downstream-heavy traffic consists mainly of Web browsing, e-mail traffic and incoming FTP traffic, which is handled well by the high downstream rate of the ADSL line. We also have the same type of traffic originating from a third server belonging to customer A. Thus, almost all the traffic from Sangoma and the third-customer server is routed through the ADSL line. The exception is outgoing SMTP mail traffic, which benefits from the increased upstream bandwidth of the Frame Relay T1 line.

Customers A and B have three servers between them. Of these, one is a Web server that needs a fixed IP address and has mostly outbound traffic. Another is a VPN server that also requires a fixed IP address; its traffic is light. All the traffic for both of these servers is routed through the T1 line with its fixed IP addressing structure.

The Sangoma policy solution is a staged process where outgoing packets traverse a set of rules and policies to achieve the desired traffic distribution. Only outgoing packets are distributed between the two interfaces, because we cannot control the path of incoming traffic. However, once the packets leave a particular interface, either T1 or ADSL, the response comes back through the same interface.

The advanced routing tools and utilities available for Linux give us the means to manage the network and achieve our desired goals. The Linux kernel supports multiple routing tables, allowing each physical connection to have its own routing table. Once we have a separate table for each of our physical interfaces, we use iptables and iproute2 to lead traffic into either routing table. From there, the packets follow a default route out to the appropriate physical interface.

The iproute2 suite contains a configuration file that is used to assign routing tables to the Linux routing stack. By default, the tr_tables contains a single routing table definition, main. This is the standard routing table used by the Linux routing stack. Listing 1 shows the routing table entry we added for our ADSL line, adsl. Individual routes are added to these routing tables using standard Linux commands. The outgoing packets must traverse six stages between router input and output.

Listing 1. Multiple Routing Tables

```
cat /etc/iproute2rt_tables
#
# reserved values
#
#255    local
#254    main
#253    default
#0      unspec
# local
```

```
#1      inr.ruhep
200 adsl
```

Input over Ethernet

The first step is iptables mangle rules where traffic is tagged as either Tag 1 for ADSL or Tag 2 for T1. To give all Sangoma mail Tag 2, for example, we apply the rule:

```
iptables -t mangle -A PREROUTING -i eth0 \
-p tcp -s xxx.xxx.xxx.82 --dport smtp -j t1_line
```

We then use the iptables --set-mark option in the t1_line chain:

```
iptables -t mangle -N t1_line
iptables -t mangle -A t1_line -j MARK --set-mark 2
iptables -t mangle -A t1_line -j ACCEPT
```

We have similar rules for traffic going to the ADSL line.

The iproute2 policy points Tag 1 to the ADSL routing table and Tag 2 to the main routing table, which goes to the T1 line:

```
ip rule del fwmark 1 table adsl
ip rule add fwmark 1 table adsl

ip rule del fwmark 2 table main
ip rule add fwmark 2 table main
```

Routing Tables

The default route of the ADSL routing table is ppp0, representing a PPP over Ethernet (PPPoE) link. The Ethernet then is encapsulated into ATM (EoA), and it is ATM cells that traverse the ADSL link to the telco DSLAM.

If the ppp0 interface goes down, the ADSL default route is removed automatically by the kernel and replaced with the main routing table. Thus, if the ADSL line fails, all traffic destined for the ADSL routing table is diverted to the presumably more reliable main routing table. We do get the occasional ADSL outages that are endemic to low cost, unmanaged broadband services like ADSL. These outages last from a few seconds to several hours, but there is no loss of user functionality because the traffic switches transparently to the T1 line. The T1 interface is good backup for the ADSL line, but the reverse is not true. Most of the hosts that use the T1 link do so because they need to use fixed IP addresses; they cannot be serviced adequately with the ADSL line that has a non-fixed IP address.

The default route of the main routing table is wan0 (T1). All traffic coming into this routing table is forwarded to the T1 line.

Masquerading Outgoing Traffic

Outgoing Internet traffic over the ADSL connection comes from servers with routable IP addresses. These address types need to be NATed; otherwise, the traffic routed to the real IP addresses comes back over the T1 line:

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Our tagging and policy routing is shown in Figure 4.

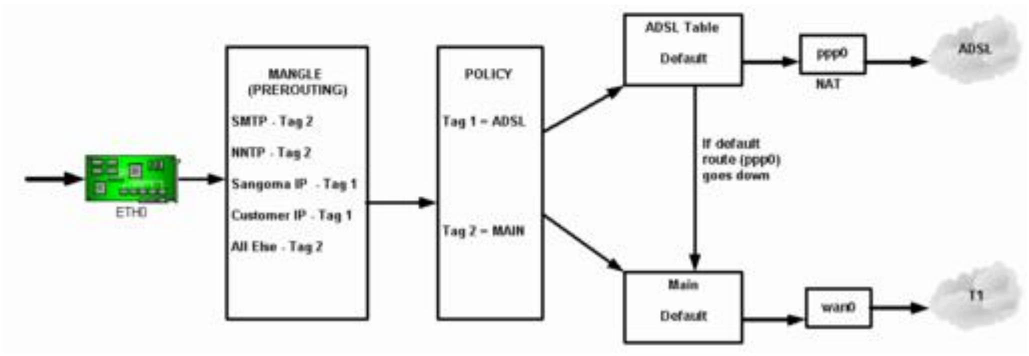


Figure 4. Tagging and policy routing allows for failover to the T1 line if the ADSL line goes down.

IP Accounting

Once we have directed the appropriate traffic to the ADSL line, we need to manage residual T1 traffic so that the usage boundaries are never exceeded. The magic 95th percentile point always must be less than or equal to 128kbps. We first measure the traffic using IP accounting, which allows us to gauge average throughput over a specified time interval.

All incoming and outgoing packets on the T1 line pass through IP accounting rules. Each customer's traffic is measured based on the IP address and direction of the traffic.

A custom dæmon has been written to check the T1 bandwidth used for each five-minute period or bin. Each time the T1 throughput is greater than 128kbps averaged over a five-minute period, a counter is incremented. The 128kbps threshold corresponds to about 4.5MB over the five-minute period.

If the counter reaches 432, that represents the free 36 hours per month. In turn, that equals 5% of the time in a month, and the TC (traffic control) script is executed to clamp the T1 line down to 128kbps, until the start of the next month. The IP accounting configuration file is shown in Listing 2, available from the *Linux Journal* FTP site [ftp.linuxjournal.com/pub/lj/listings/issue121/7134.tgz].

Traffic Control

We usually get through the month without having to clamp the T1 line. Sometimes, however, the free 36 hours are consumed. In this case, traffic control (TC) is used to clamp the bandwidth. The documentation covering traffic control and the tc command can be found at lartc.org/manpages.

We use Qdisc class-based queuing (CBQ) discipline for both the T1 line, wan0 and the local Ethernet, eth0. This is done for both connections to implement flow control in both traffic directions:

```
tc qdisc add dev wan0 root handle 10: \
cbq bandwidth 1500Kbit avpkt 1000
tc qdisc add dev eth0 root handle 20: \
cbq bandwidth 1500Kbit avpkt 1000
```

Next, add Global Class with maximum bandwidth for wan0 and eth0. The maximum bandwidth for both lines is 1,500kbps (T1):

```
tc class add dev wan0 parent 10:0 classid 10:1 \
cbq bandwidth 1500Kbit avpkt 1000 rate 1500Kbit \
allot 1514 weight 150Kbit prio 8 maxburst 0
tc class add dev wan0 parent 20:0 classid 20:1 \
cbq bandwidth 1500Kbit avpkt 1000 rate 1500Kbit \
allot 1514 weight 150Kbit prio 8 maxburst 0
```

Add User Class with limited bandwidth for both wan0 and eth0. The bandwidth limit we use is 100kbps, not 128kbps. Linux TC is not perfectly accurate, and we determined through trial and error that if we set the bandwidth limit higher than 100kbps, sometimes the burst traffic could go over 128kbps:

```
tc class add dev wan0 parent 10:1 classid 10:100 \
cbq bandwidth 1500Kbit avpkt 1000 rate 100Kbit \
allot 1514 weight 10Kbit prio 8 maxburst 0 bounded
tc class add dev eth0 parent 20:1 classid 20:100 \
cbq bandwidth 1500Kbit avpkt 1000 rate 100Kbit \
allot 1514 weight 10Kbit prio 8 maxburst 0 bounded
```

Add SFQ queuing discipline for the User Class, on both wan0 and eth0. The default queuing discipline Stochastic Fairness Queueing (SFQ) has been selected. A number of other disciplines also could be employed:

```
tc qdisc add dev wan0 parent 10:100 \
sfq quantum 1514b perturb 15
tc qdisc add dev eth0 parent 20:100 \
sfq quantum 1514b perturb 15
```

Bind the traffic tagged number 2 to the User Class Queue for both wan0 and eth0. All traffic destined for the T1 line already has been tagged with number 2. The traffic control only limits the T1 traffic, while letting ADSL traffic flow at its full physical rate:

```
tc filter add dev wan0 parent 10:0 protocol ip \
prio 25 handle 2 fw flowid 10:100
tc filter add dev eth0 parent 20:0 protocol ip \
prio 25 handle 2 fw flowid 20:100
```

Results

The policy routing works perfectly as programmed, directing the traffic as appropriate to the T1 and ADSL links and providing redundancy in case the ADSL link fails. The traffic management on the T1 has been satisfactory, and we generally have been able to provide our users with a respectable service transparently. Of course, the consistency of traffic throughput during a single month is dependent on how rapidly the free bandwidth is consumed.

As an example of our T1 traffic management see Figure 5, which shows Frame Relay T1 bandwidth usage during May 2003. The red line on the graph represents 128kbps, which is our threshold limit for billing. Throughput clamping occurred after May 23. One of our customer's servers became infected with a virus that generated a great deal of traffic during the month, consuming our precious free bandwidth. As a result, these customers were required to exist for more than a week running at 128kbps on the T1 line. ADSL traffic, of course, was not affected.

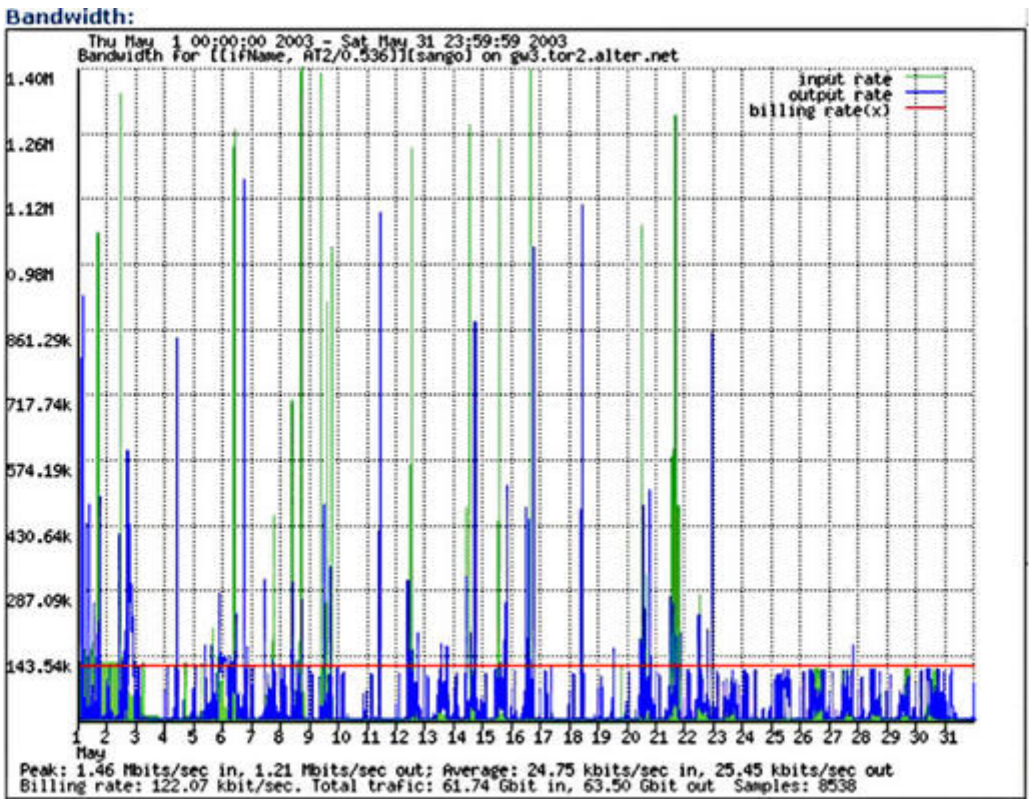


Figure 5. Bandwidth Usage by Five-Minute Bins during May 2003

The same data presented with the five-minute bins listed by bandwidth is shown in Figure 6. This graph may be compared with the ideal usage shown in Figure 3. Notice the billing rate of 122.07kbps indicated in this figure. This reflects the success of the traffic control procedures in ensuring that the billing rate remained below 128kbps.

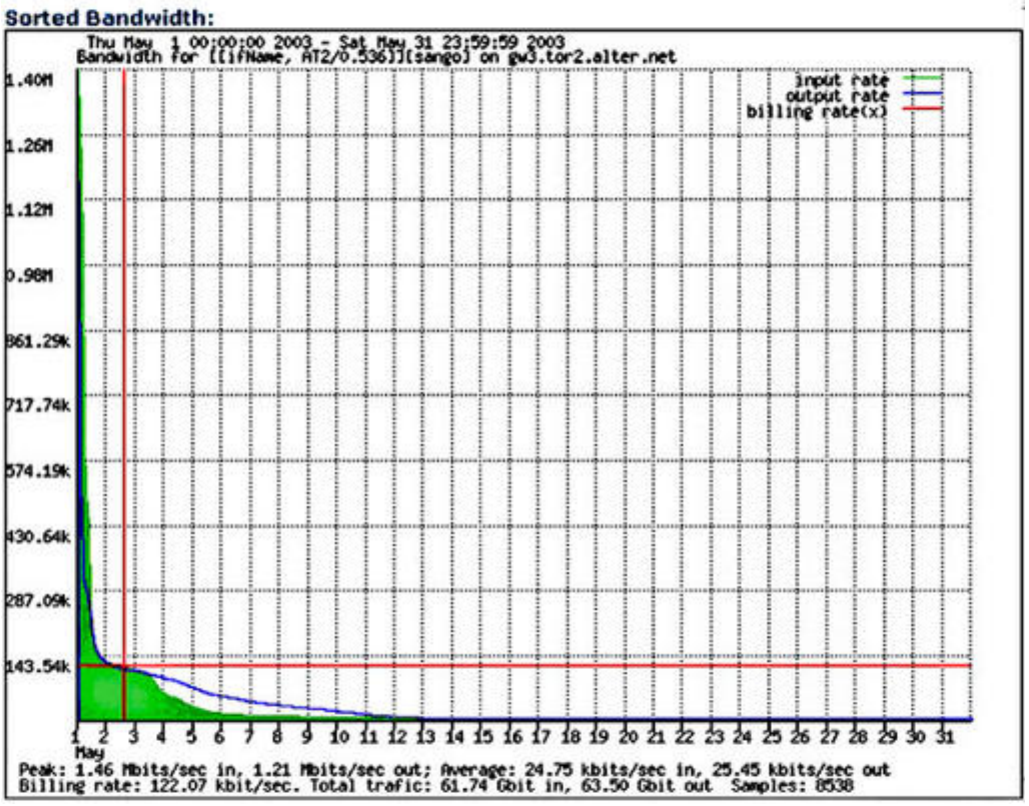


Figure 6. Bandwidth Usage by Five-Minute Bins during May 2003, Ranked by Bandwidth

Conclusion

Although this is quite a simple implementation of policy routing, IP accounting and traffic shaping, it does provide an illustration of how the Linux advanced routing tools can provide the kind of control needed to manage sophisticated traffic policies.

David Mandelstam is President of Sangoma Technologies Corp. Founded in 1984, Sangoma develops and manufactures wide area network (WAN) communication hardware and software products, with an emphasis on the PC platform. The communications solutions and routing products support all popular WAN networks, line protocols and all standard PC operating systems and platforms.

Nenad Corbic is Senior Linux Developer at Sangoma Technologies Corp. (www.sangoma.com).

Archive Index Issue Table of Contents

Advanced search

The Linux-Based Recording Studio

Aaron Trumm

Issue #121, May 2004

With a Linux-based hard disk recorder, you can create your own project studio on a budget. Now the only thing between you and that great album you want to make is practice, man, practice.

I grew up using keyboards. Cold-war grey TRS-80s, green-screened Apple IIs, IBM clones, 8088s, 286s, PC-DOS, then Windows (missing the command line) and finally UNIX command lines. Later, the recording bug bit me and took me away from the command line and into studios. I still was a PC guy, but there never was a reason to bring a computer into the studio. Affordable hard drives and memory were too small for audio, sound cards were junk and processors were too slow.

Then, Linux came along. Sure, I had to wait for hard drives to get bigger and chip speeds to increase, but even after that, proprietary software still was way out of reach. So I upgraded my studio, learning a lot about Linux along the way. Here, I share a bit of what I did in my studio and explain how you might start a Linux-based studio. General information about Linux audio and recording is vast, so I refer you to further resources where appropriate (see the on-line Resources section).

How to Set Up a Linux Studio

Like anything, what you need to buy for your studio and how you set it up is determined by a few key decisions, especially when it comes to studio hardware. The hardware is easy as 3.1415. Anything that runs Linux can run Linux audio applications, but bear the following in mind:

- Audio uses 5MB per track minute at CD quality (44.1KHz, 16 bit), meaning a three-minute song recorded in stereo takes up 30MB on the hard drive. Multitracking uses more than two tracks. A typical project of 24 tracks that is three-minutes long would use 360MB, not including captured audio being used.

- Slight upgrades to things like RAM size and CD-ROM speed are nice if you have older equipment. A CD writer is your friend, too, as you might have guessed.
- Some bad video cards introduce noise into the sound card.
- Drivers in Linux are sometimes hard to come by, so read and ask around before buying hardware, especially sound cards.

Acquiring software is almost as easy. Latency needs to be low, so the kernel needs a bit of a tweak in the form of a low-latency patch. The hard drive needs to be tuned correctly too. This subject is more than I can cover here, but check out the Resources on the Web for other readings. Also, keep a dual-boot system with Microsoft Windows for troubleshooting. You may need to test hardware on another operating system to narrow a problem to a Linux driver, or you may have tasks, such as upgrading firmware, that need to be done on a Windows box.

Now that we've got the box, it's time to decide what studio hardware we need. I like to think of the signal flow for a given project, and that tells me what I need. Figure 1 shows the basic concept of where a signal goes in a recording project. Also take a look at Figures 2 and 3; 2 is a wiring scheme for a simple studio and 3 shows my studio's scheme. I begin with the lynchpin, which actually is a couple of rungs down on the signal chain.

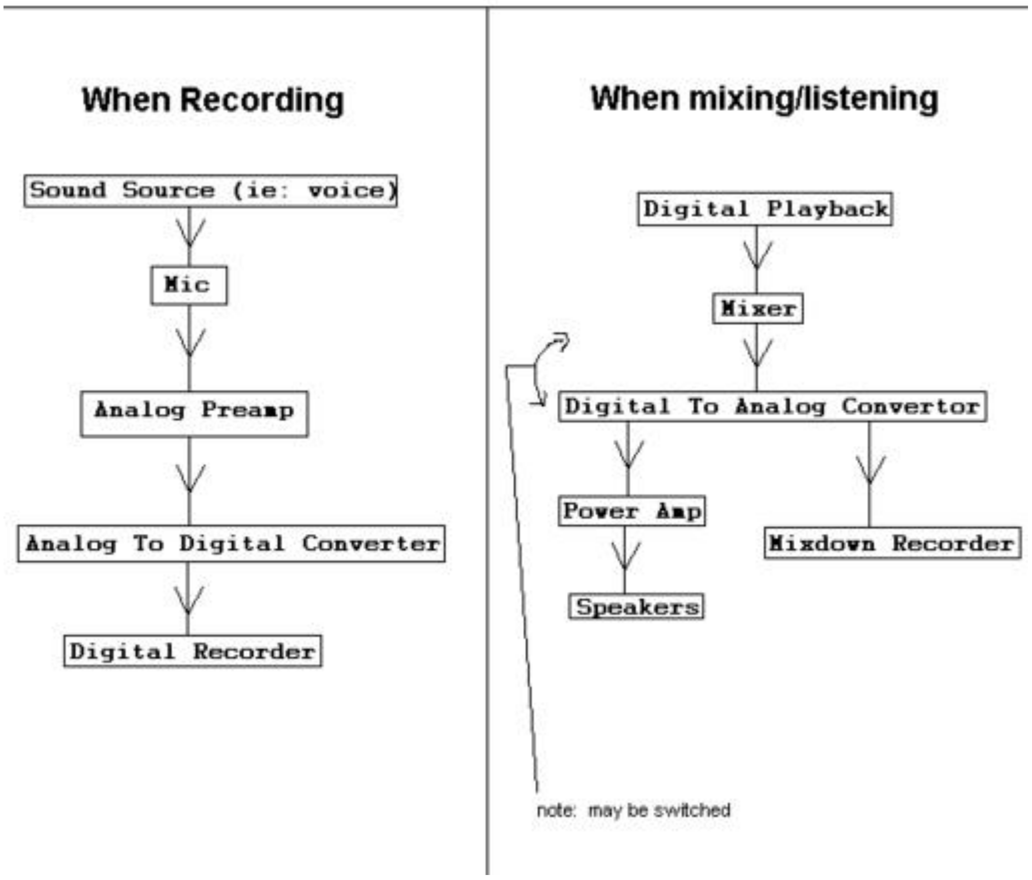


Figure 1. Basic Recording Signal Flow for a Simple Project

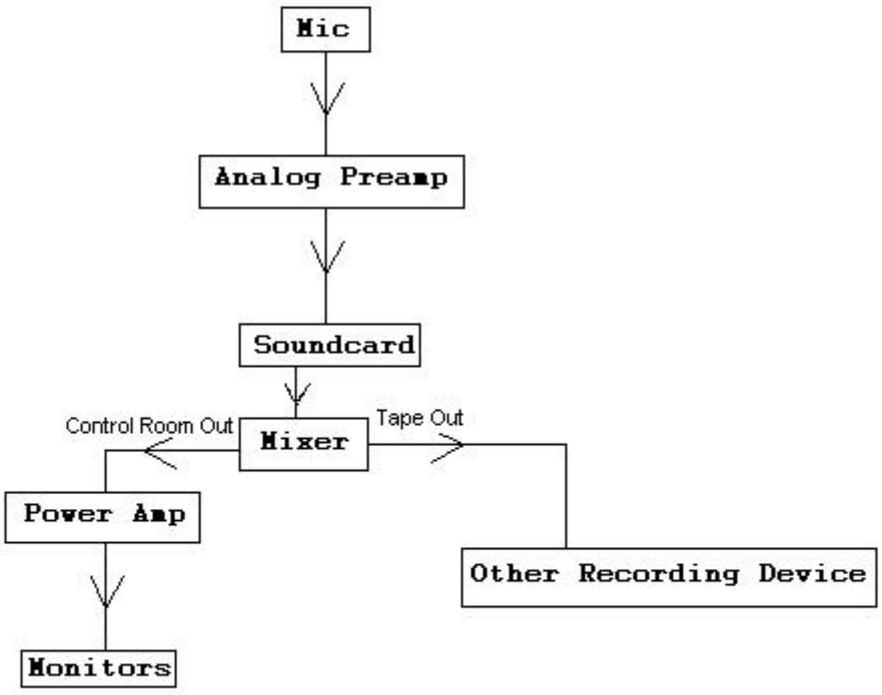


Figure 2. A Simple One-Way Signal Flow in the Studio

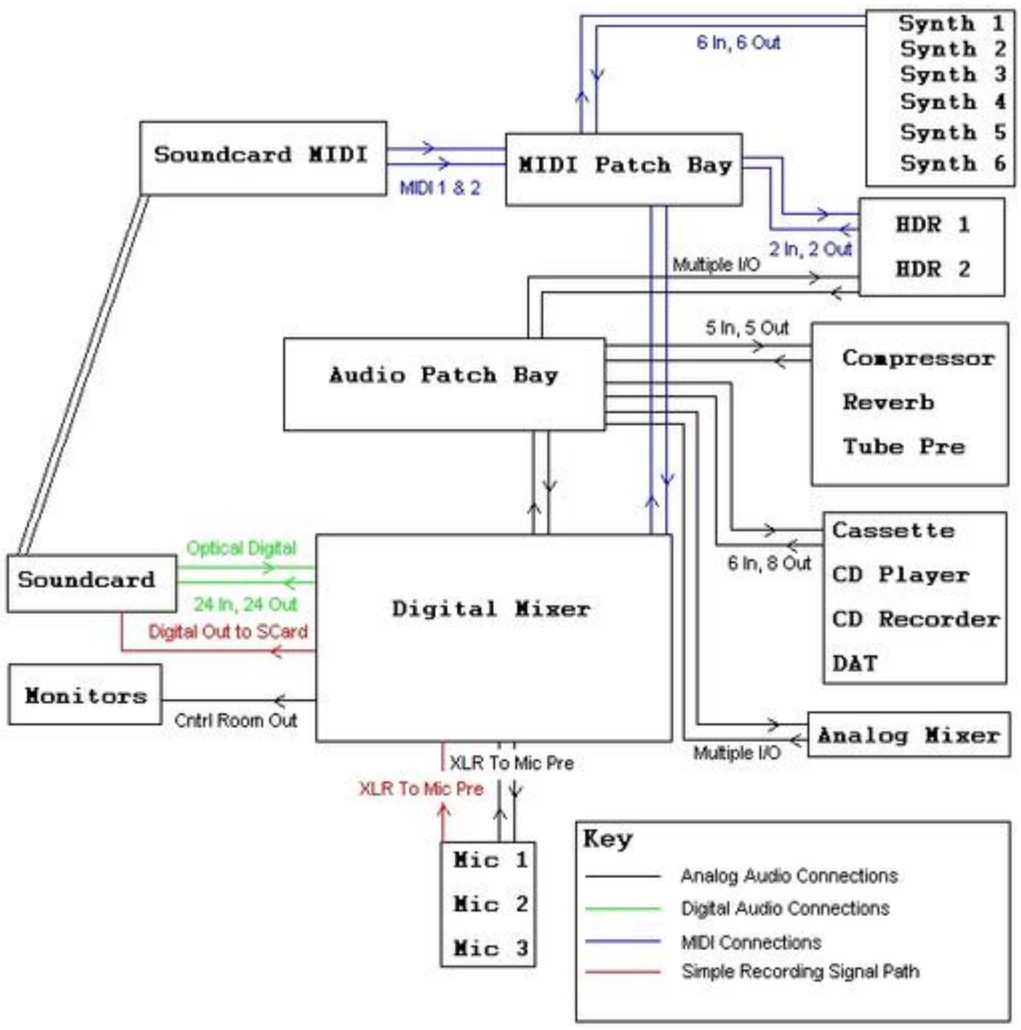


Figure 3. The Author's Studio

Analog-to-Digital Conversion

The key to digital recording is analog-to-digital and digital-to-analog converters (ADCs and DACs). In other words, you need to get sound in to and out of your computer. In both directions, you have some decisions to make.

ADC must be done in order to record. This happens in the sound card, in a digital mixer or in a standalone ADC.

Getting sound out (DAC) consists of two parts, listening (or monitoring—more on that below) and mixing. When mixing, you might never convert back to analog. You might mix digitally inside or outside the computer, saving a mix as a .wav file or transferring digitally to a digital recorder. The thing to understand here is that at some point before you can hear it, a DAC must happen. If you've done everything digitally, make a CD and play it in your car, that's where the DAC has happened.

The Mixer: Analog, Digital or Software?

As you can imagine, your choices are endless. You conceivably could choose any combination, such as converting analog to digital with the sound card while converting digital to analog outside, or vice versa. It's a bit easier, though, to pick one place to do all your conversions, either in the sound card or somewhere else. If you simplify it this way, your decision comes down to whether to have an external mixer.

In the simplest configuration, say if you were using a consumer sound card with only one stereo output, you'd mix entirely in the computer and not think much about the ADC and DAC being done in the sound card. You would then have no external mixer (see the Preamps section below).

If you do have an external mixer, you're either using an analog mixer or a digital mixer. In either case, you need a professional sound card that can separate channels, as opposed to a consumer card that outputs only one stereo mix, requiring you to mix inside the computer.

If using an analog mixer, your DAC and ADC happen in your sound card. Therefore, you need a sound card or sound card/breakout box that has analog inputs and outputs, such as RME's PCI cardbus and Multiface combination card. There is a nice primer on this card on the *LJ* Web site (see the on-line Resources).

If you choose a digital mixer, your DAC and ADC happens in the mixer itself. You therefore need a sound card that has digital inputs and outputs that are compatible with your mixer. In my case, this is RME's HDSP 9652.

Many digital mixers have built-in effects and processing, including reverb, compression and noise gates, as do many software packages. Few analog mixers offer such features, so if you're doing traditional analog mixing, you might spend more on outboard effects and processing. There still are plenty of reasons to use these tools if you've got the money, but on a budget, I recommend a digital mixer.

A few questions to ask about potential sound cards:

- Is it noisy?
- Does it have the ability to record while playing back (duplex mode)?
- How many channels can it play back at once?
- How many channels can it record at once?
- What kind of physical I/O ports does it have?
- Does it have built-in MIDI?

- Is there a Linux driver for it?

A great place to answer the last question is the ALSA Sound Card Matrix (see Resources).

Microphones

If you're recording acoustic sources, such as voice or drums, you need microphones. Your budget and what you're recording influence your decisions here. For example, if you have a medium to large budget and need to record an acoustic guitar and singer, I might recommend two AKG 414s (about $1,000 US each). If you need to record pristine vocal tracks and have a large budget, I might recommend one Neumann U87 (about $3,000 US). Or, maybe you have a small budget but still need to record vocals and an electric guitar amp. Then I might go with a pair of Shure SM58s, about $100 US each. Of course, if you never record acoustic sources and only use synths plugged in directly, you don't need microphones or preamps.

Preamps

The signal from a microphone needs to be amplified before it is loud enough to record or broadcast properly. If you plug a computer microphone in to the microphone input of a consumer sound card, you're using a preamp, and you should get a loud enough signal. If you try plugging in to the line input, you barely get anything. Professional sound cards don't have 1/8" microphone inputs and assume you have outboard preamps.

The question is whether to use standalone preamps or the preamps built into a mixer. For most, the preamps in almost any mixer are sufficient. The only reasons not to use the mixer's preamps are if you don't have a mixer, you need more at once than your mixer has or you have aesthetic reasons to use a standalone.

The need for preamps is a good case for having an external mixer, because having a professional sound card with multiple analog inputs, no mixer and a bunch of outboard preamps usually is more costly and less flexible than having a mixer.

Monitoring

To listen, you can use anything you please, from computer speakers to headphones to a home stereo speaker/amp combination to studio reference monitors. There is, however, a distinction between speakers and monitors. A speaker is designed to enhance the sound of a recording, and a monitor is designed to give an accurate, uncolored representation. If you want to do accurate work, you need monitors.

Studio monitors come in a variety of flavors. The major thing you need to know is whether a set is powered. If it isn't, you need a power amp, just as if you were using a set of home stereo speakers. Replacing your regular speakers with studio monitors and connecting to your existing amp is easy.

Digital Recorder

The Linux box is your digital recorder. The decision to make here concerns software. Literally hundreds of open-source audio applications are available for Linux, from hard-disk recorders to MIDI sequencers to MP3 encoders. I don't have room to talk about them all, so I focus on my main studio tool, Ardour. (See the Where to Start section of the Resources page on the Web for more information on finding software.)

You can Google your way to most software, but there are some great package resources out there. I'm on Red Hat, so I use Planet CCRMA. The Planet is a project at Stanford's Center for Computer Research in Music and Acoustics, maintained by a knowledgeable guy named Fernando. Not only does Nando maintain Red Hat RPMs of most audio and video applications, drivers, utilities and even custom kernels, he has an extensive guide for installing kernels, ALSA sound drivers and software, as well as for tweaking your machine's performance. I highly recommend reading through the Planet, even if you're not using Red Hat. There are other similar resources for different distributions.

To quote the Ardour home page, “Ardour is a multichannel hard-disk recorder (HDR) and digital audio workstation (DAW). It is capable of simultaneous recording of 24 or more channels of 32-bit audio at 48KHz....” Ardour needs a 2.4 or later low-latency kernel, 0.9 series or later ALSA sound drivers and JACK (Jack Audio Connection Kit). It also needs a window manager because it doesn't run from the command line like many other Linux audio applications. I run Ardour from Fluxbox and sometimes KDE, but most managers should work.

Ardour should be fine with any sound card supported by ALSA. Part of why I use the HDSP is because Ardour was written with RME's cards in mind. Ardour looks and acts a lot like Pro Tools from Digidesign.

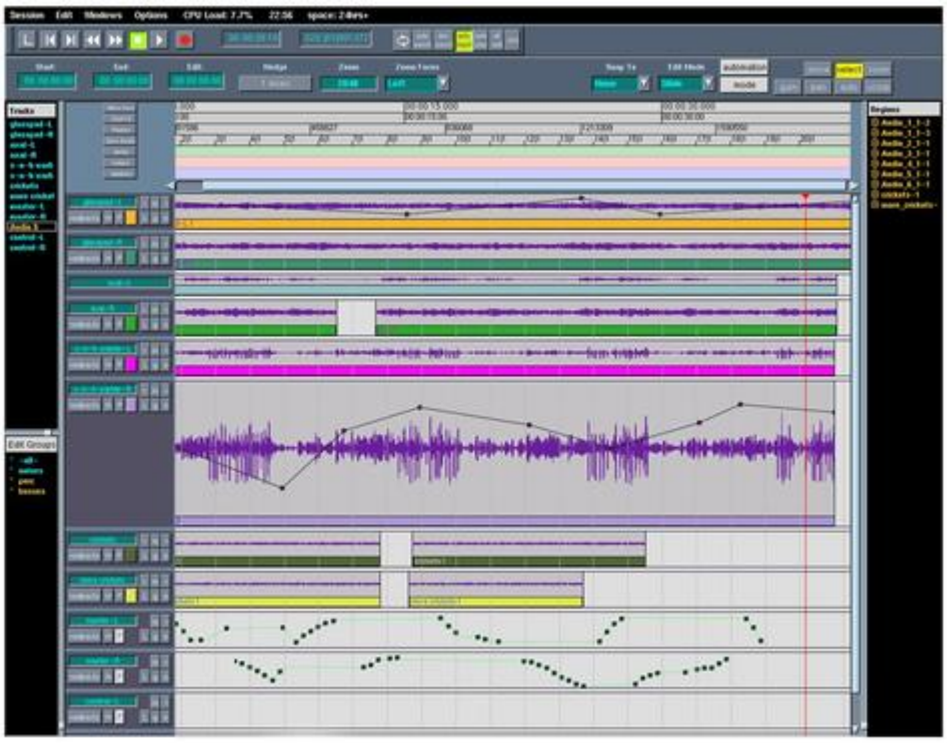


Figure 4. The Ardour Edit Window

Starting Ardour is a matter of starting JACK and then starting Ardour while JACK runs. It's best to run these as the superuser, because only root is allowed to invoke real-time priority. A generic start command for JACK would be:

```
jackd -d alsa -d hw:0
```

This starts the JACK server using ALSA as its device, and the default sound card as ALSA's device. See the JACK User Documentation to learn more about command-line options for JACK.

Like Pro Tools, Ardour is very powerful. You can create as many audio tracks as your hardware can handle, record tracks, mix internally, apply plugins and route them any way you and your sound card can imagine. A typical session for me might see 20 Ardour tracks routed to 20 separate card outputs, and eight more tracks submixed within Ardour and sent to two more channels of output, all mixed on my digital mixer. It's relatively easy to do this. I simply click on the Out button, toward the bottom of each track in the mix window (Figure 5), and choose an output channel from a pop-up list.

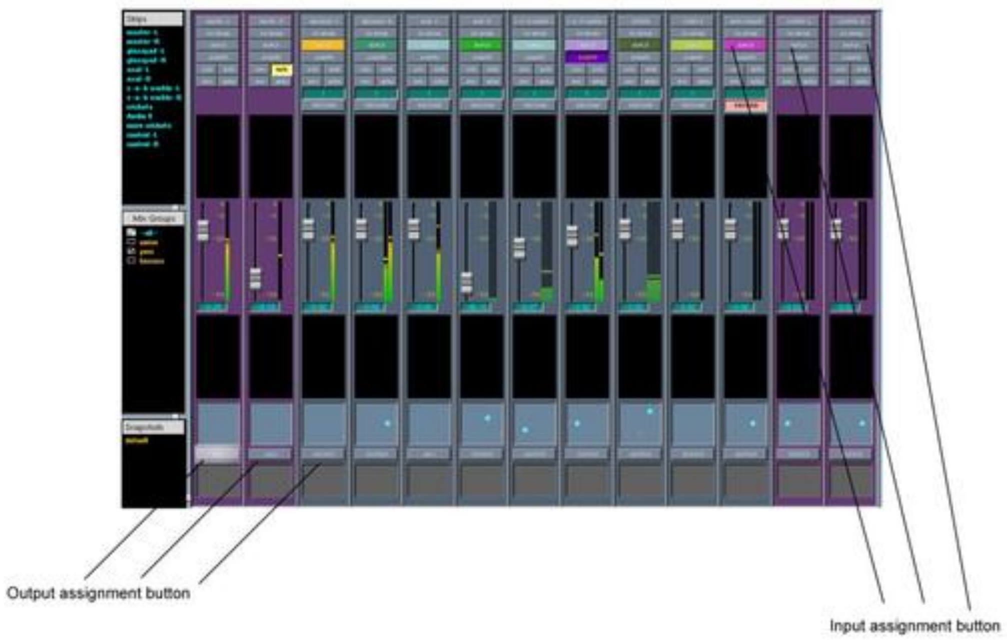


Figure 5. The Ardour mix window lets you select the input, output and level for each channel.

Another option is mixing totally within Ardour and exporting the session as a .wav file. The mix window has graphical faders, exactly like Pro Tools, as well as plugins and automation. Automation is as simple as clicking arec, moving your settings, then unclicking arec and clicking aplay to play back the automation.

As you can see, using Ardour is as straightforward as any professional DAW, which isn't totally straightforward, but it doesn't take long to learn. Because it's in beta, the manual is forthcoming, although a read through the Pro Tools manual should provide a good idea of how it works. There also are some good HOWTOs on-line (see Resources). At the time of this writing, Ardour is at 0.9beta8-1. It's important to keep this in mind, save often and don't be alarmed by the occasional crash. You can help get it to version 1 by reporting bugs (see Resources).

Space

Studios consist of some combination of control room, recording space and isolation rooms. If you've got the space, you can have all of them; if not, you may be limited to only your control room. Figure 6 is a typical studio floor plan; Figure 7 is my studio's floor plan.

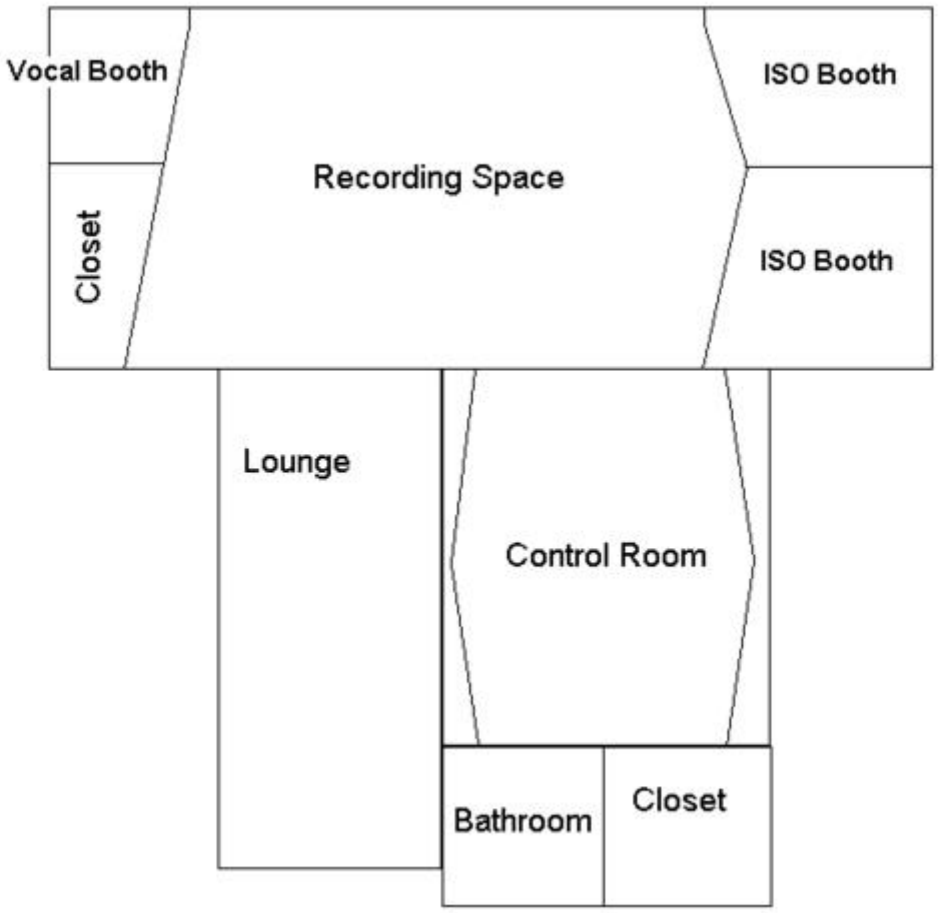


Figure 6. A typical studio floor plan includes isolation (ISO) booths separate from the main recording space, but it can be simpler.

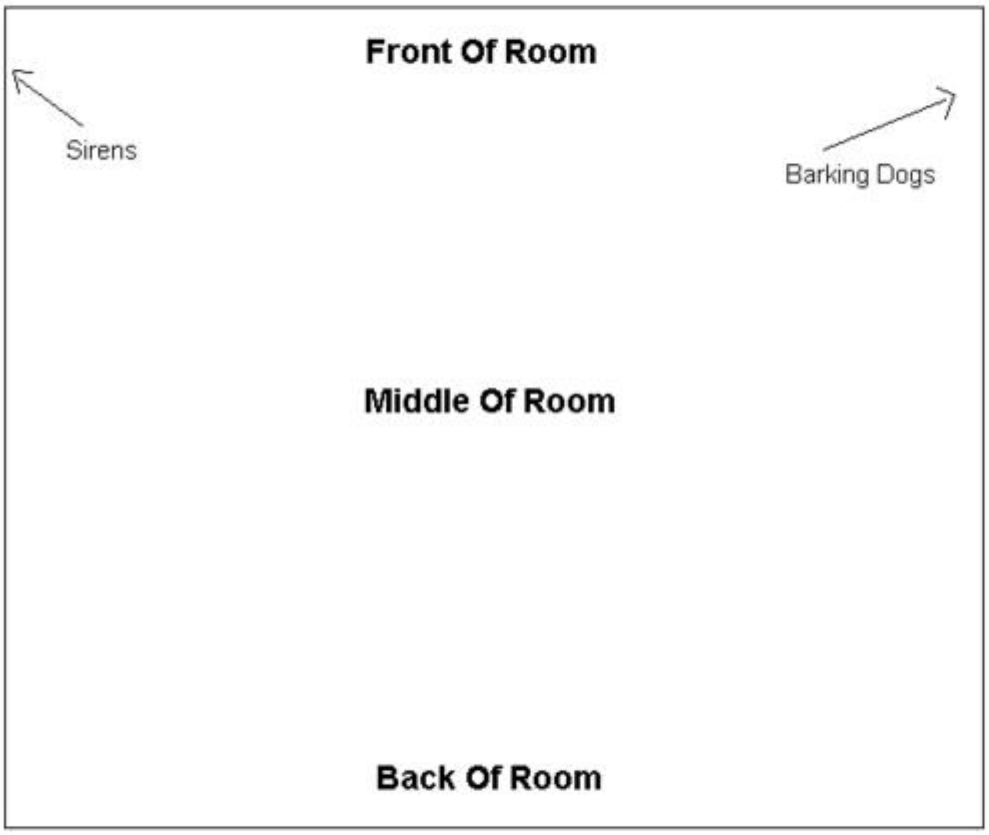


Figure 7. The Author's Super-Fabulous, Ultra-Creative Studio Floor Plan

Some people get expensive rigs, put them in an office and call it a professional studio, which is far from the truth. The best thing you can do to improve your recordings, better than buying $5,000 US microphones, better even than 77-string custom guitars made by Beelzebub himself, is improve your studio's acoustics. There are two areas to consider, recording space and listening environment. It's easy to be off the mark with your recording space and easier still to be dead wrong in your listening space.

You should find some information about bass traps, no parallel surface rule, diffusion, absorption, isolation, flutter echo, reverb times and the like on the Resources page on the Web. Then you can start deciding things like where to place furniture and acoustic material, finding a good room that's not a hallway next to a jackhammer and so on.

Practice

Good studio practice is more than a computer and its fancy open-source applications. For example, don't forget to take tracks outside of the computer's domain. You may want to use a tube preamp, a classic reverb or an outboard compressor. Experiment, don't be afraid to fuse the old with the new and admit when your software isn't giving you what you really want. A drum machine never can replace a drummer.

You also want to be wise about your cabling and general studio maintenance. Keep audio cables away from AC cables, cross them only at right angles when absolutely necessary and keep your connections clean. See the Resources page on the Web for some general recording information that should be helpful.

There you have it, the fusion of computer geek and recording nerd. You're now a few steps closer to your Linux-based studio. When you need help, check out the Mailing Lists on the Resources page. Good luck, and raise your glass to some ingenious open-source records appearing in stores everywhere.

Resources for this article: /article/7457.

Aaron Trumm started recording pause loop tape hip-hop at 14. He has since released seven albums and countless side projects. He created and still owns NQuit Records, and he formed the Techno/Classical/Poetry Project Third Option, which includes his classical piano improvisation and poems, as well as poetry from Tamara Nicholl, who was the first ever female Albuquerque City Poetry Slam Champion. Aaron was also the tenth-ranked slam poet in the US in 2002 and has competed at the National Poetry Slam four times.

Using SQL-Ledger for Your Business

David A. Bandel

Issue #121, May 2004

If you're keeping a proprietary OS around just to run the accounting software, you're missing a chance to step up to the flexible, better-supported alternative.

Back in the late 1990s, I started looking for a good accounting software package for Linux. I was disappointed by all the offerings I found; none was up to snuff or even looked like it might ever be. Then, about three and a half years ago, I stumbled on SQL-Ledger (SL) written by Dieter Simader (see the “Making Open Source Work” sidebar).

Making Open Source Work

Dieter Simader is the author of SQL-Ledger. He started the project four years ago. His software has taken off since then, and he now works on SL full-time. He's made his open-source project, released under the GPL with no cost to others to use, his primary source of income.

So how does one make a living off free software? If you visit the SL site, you can see that Dieter offers support contracts, which are all reasonably priced. His software is of high quality, and his offerings include data conversion, pay-as-you-play support and support for the development version if you need the more advanced features. He also sells the software manual that tells you not only how to use his program but how to customize screens for all users or for certain users, using SL's API.

These value-added features are well worth the price and should be considered essential for any business. Although businesses can hire someone else to do what Dieter does, including supporting the software, customizing templates and importing data, few could do the work as well as Dieter.

As a consultant, my clients usually want to deal with me, not someone on another continent whom they don't know. For that convenience, they are willing to pay a premium. So I tack a small percentage on to Dieter's price, pay Dieter and we're both happy. He gets work and is paid for supporting his program. I get an annoyance fee, because my clients will annoy me if the software doesn't work to their satisfaction. This model also serves as a good way to expand into markets where the author doesn't speak the language, so additional revenue also serves as a translation fee. SQL-Ledger is proof that open-source business models can work.

To be honest, at first glance I thought SQL-Ledger wasn't yet an offering for a serious business. It lacked point of sale (POS), payroll and a number of other features. But, based on its ease of installation, its flexibility and a number of other factors, not the least of which was the author's rather ambitious to-do list, I tried it out.

I used SQL-Ledger for my IT consulting business, Pananix, SA, and it proved to be more than adequate. I could input customers and vendors, create invoices and orders, and print basic reports, including trial balance, income statement and balance sheet. SL even had rudimentary support for goods and services reports covering product inventory and services.

Because I currently live in a Spanish-speaking country, I anticipated problems. I'm not familiar with Spanish accounting terms, but the laws of this country require that the program interface and all statements and reports be in Spanish for the natives and tax auditors. Fortunately, SL is written in such a way that users can choose the language they require. That means I can see everything in English, while my accountant and the local tax authorities see Spanish—perfectly legal. If a language doesn't exist, it can be added easily. This feature was not well supported in any other accounting package I looked at and was a major factor for me in choosing SL.

Installation

Installation was a breeze back when I first looked at SL, but it's even easier now. The biggest stumbling block for most people involves the few requisite Perl modules for database support. A second stumbling block comes with the configuration of PostgreSQL itself, but all these are covered in the instructions and FAQ. As long as they are followed to the letter, even newbies shouldn't have any difficulty. The problems seem to arise when folks wander off on their own and deliberately or accidentally make changes affecting permissions, occasionally permissions on the database itself.

Once Perl and whatever supported database you want to use are set up, the rest is easy. MySQL is not supported and will be only when it provides certain parts of the SQL-92 standard it is now missing. PostgreSQL is the database used for design and testing, but Oracle and DB2 also work.

You also need to configure your Web server of choice to access SL, but that involves only copying a few lines to httpd.conf and restarting it. If you install SL under your DocumentRoot, even this step can be omitted, as long as you can run CGI scripts from below your DocumentRoot. If you want to print reports to disk as PDF files, you need to make sure you have LaTeX installed.

The currently recommended installation practice for SL itself is to use the author's supplied setup.pl file to handle everything for you. This makes it difficult to go wrong, and the script also is used to upgrade SL. The author recently included code to test the database itself during upgrades to ensure you don't have version mismatches between the database and the code. If you do, it automatically upgrades it (nice touch, that).

Configuration and Security

Those of you who follow my writings know that I consider security to be job number one. Accountants out there should be pleased to know that security in SL can be implemented on a user-by-user basis. Therefore, one user can see only Accounts Payable while another can see only Accounts Receivable.

SL also can be configured to comply with generally accepted accounting principles (GAAP). Most countries have their own version of GAAP, but these practices are similar. You therefore can configure SL so users can't go back and delete transactions but must post reversing entries instead. You also can close periods so nothing can be edited in prior periods.

All this is accomplished in the administration section where you add users and basically tell the system how you want it to act. After that, it's up to you. The FAQ contains information to assist you in tightening security on your system. Basically, you decide how secure or open your system will be.

Running SQL-Ledger

SL is easy to use and fairly simple to customize in any way you need. Everything in SL revolves around its Chart of Accounts. When you set up SL, you choose one to load. But making changes to that one or even creating a new one is not difficult. In fact, many businesses probably will want to sit down and make some modifications.

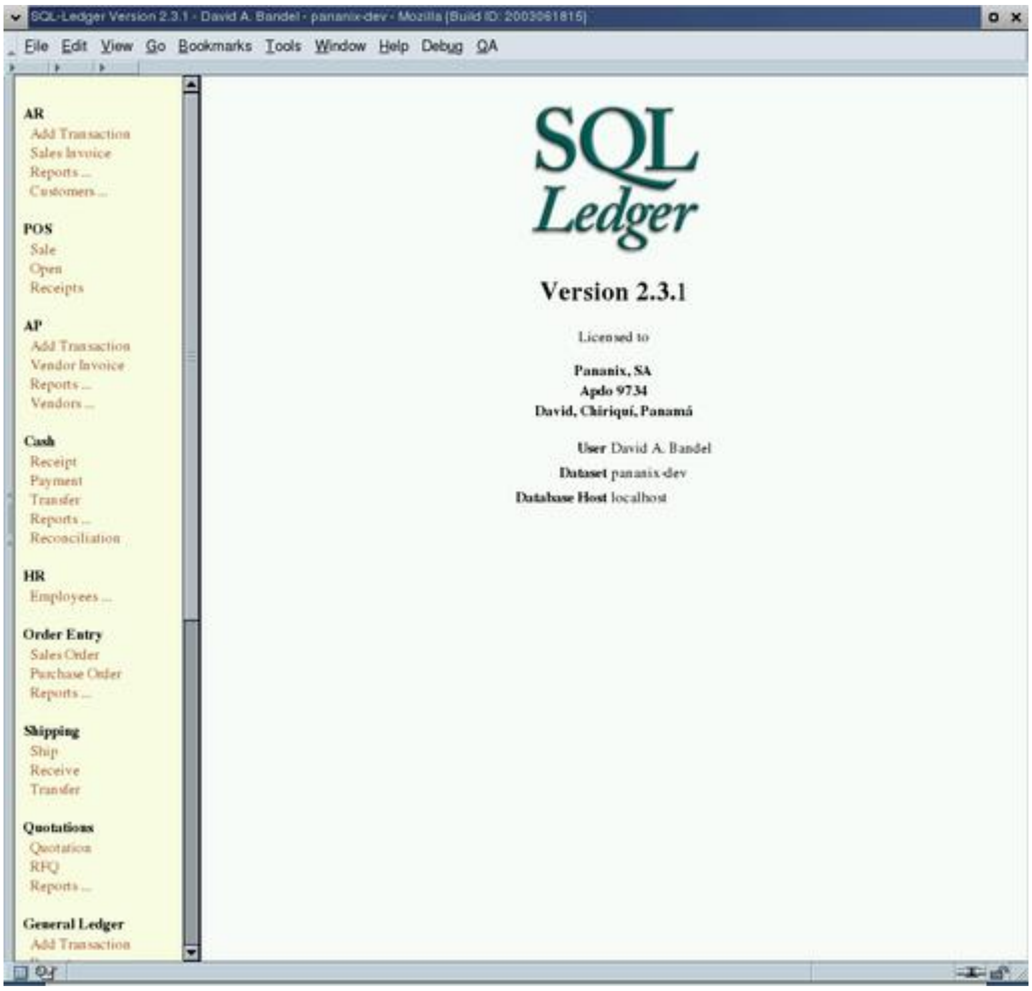
The way SL's tax system is set up, almost any tax system can be configured easily, simply by linking from one table to another using the tax percentage. The ease with which this can be done makes it ideal for locations where the tax structure might require two or three separate taxes be applied to a sale. The Default Chart of Accounts is set up with three tax accounts just to show how it is done. Tax tables contain multiple links to customers, vendors, parts and services, and a match determines whether tax is applied or not. AR and AP are independent from one another, and in combination with tax settings for customers, vendors' parts and service, you have a very flexible model to calculate tax. You even can set up negative taxes to calculate tax withholdings at source or tax on tax. The tax system here requires that I charge taxes on services tied to the sale of a taxable item; otherwise, I don't charge tax on purely a service offering. So I had to create a service call that was nontaxable and one that was taxable.

So that customers don't notice that I sometimes charge tax on a service but other times I don't, and because the taxable service always was tied to a hardware sale, I simply created an installation package that included the hardware and taxable service with the entire bundle being taxed. I haven't seen a single other accounting package for Linux that offers me this kind of flexibility.

SL can be accessed from any system with a Web browser, text or graphical, from anywhere you can reach the SL server by HTTP or HTTPS. If, like most people, you're using a graphical browser, after login you see two frames. The one on the left contains a menu broken down into several sections with items below them, and the one on the right contains a main screen where you can enter data.

One of the first things I do after each upgrade is `cd` into the bin/mozilla directory and edit menu.pl to widen the menubar. For me, it's a little too narrow, and making it about 35 pixels wider makes it more pleasing to my eyes. For those of you who use a text browser, like Lynx, that doesn't render frames, the menu headings are at the bottom of the page.

The major headings that show up depend on the user's configuration from the admin page. Entire menu headings can be removed or only specific items. So any given user's menubar may look a bit sparser than that shown in the screen captures, depending on setup and version in use. The screenshot in Figure 1 clearly shows this is Version 2.3.1, a development version. It is slightly more feature-rich than the stable version, but its designation as unstable warns you it hasn't been as thoroughly tested as the stable version. Major menu headings include AR, POS, AP, Cash, HR, Order Entry, Shipping, Quotations, General Ledger, Goods & Services, Projects and Reports.



Version 2.3.1

Licensed to

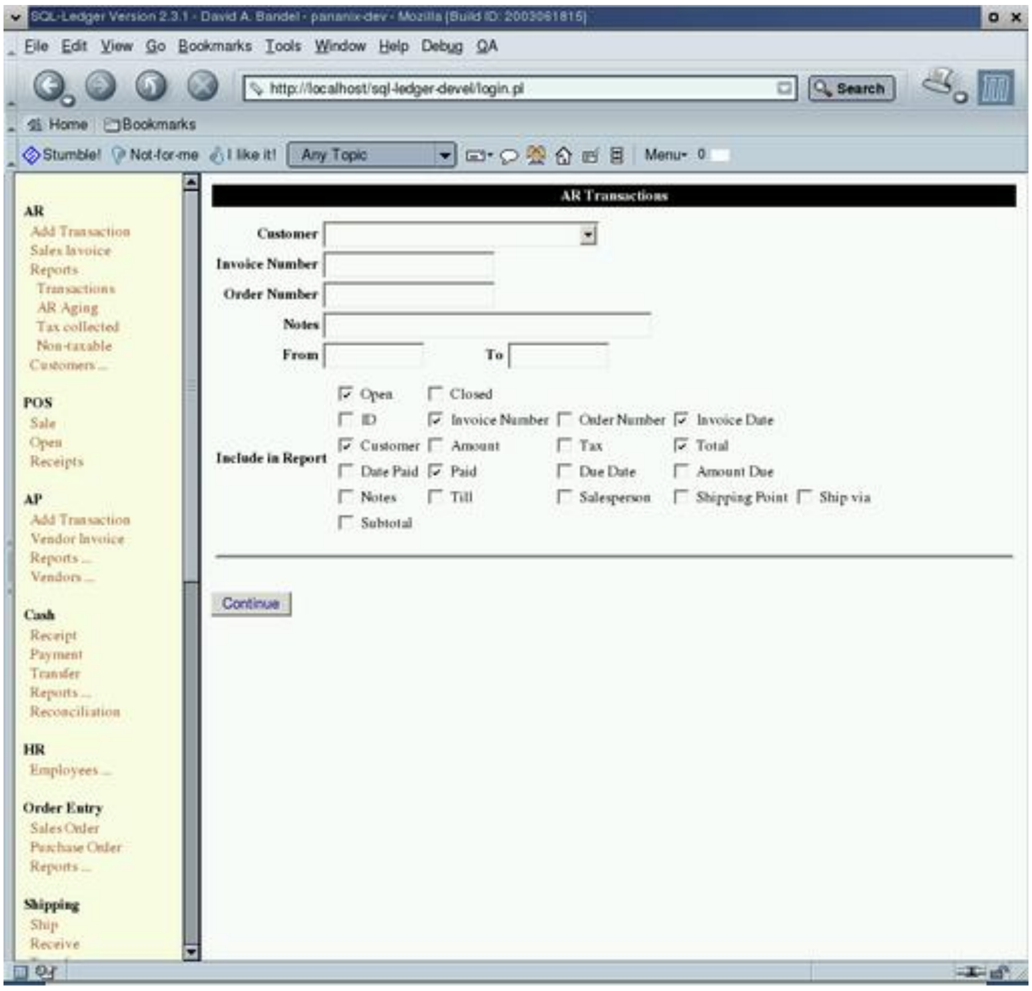
Pananix, SA
Apdo 9734
David, Chiriquí, Panamá

User David A. Bandel
Dataset pananix-dev
Database Host localhost

Figure 1. SQL-Ledger Start Page

You also can see some menu items followed by ellipses. Those bring up even more detailed submenu items. In the case of System..., a long list is brought up in later versions.

Taking a quick look at AR Reports, selecting Reports expands the menu list. Then selecting Transactions provides a screen to define the transactions we want to see and the information we want presented; see Figure 2.

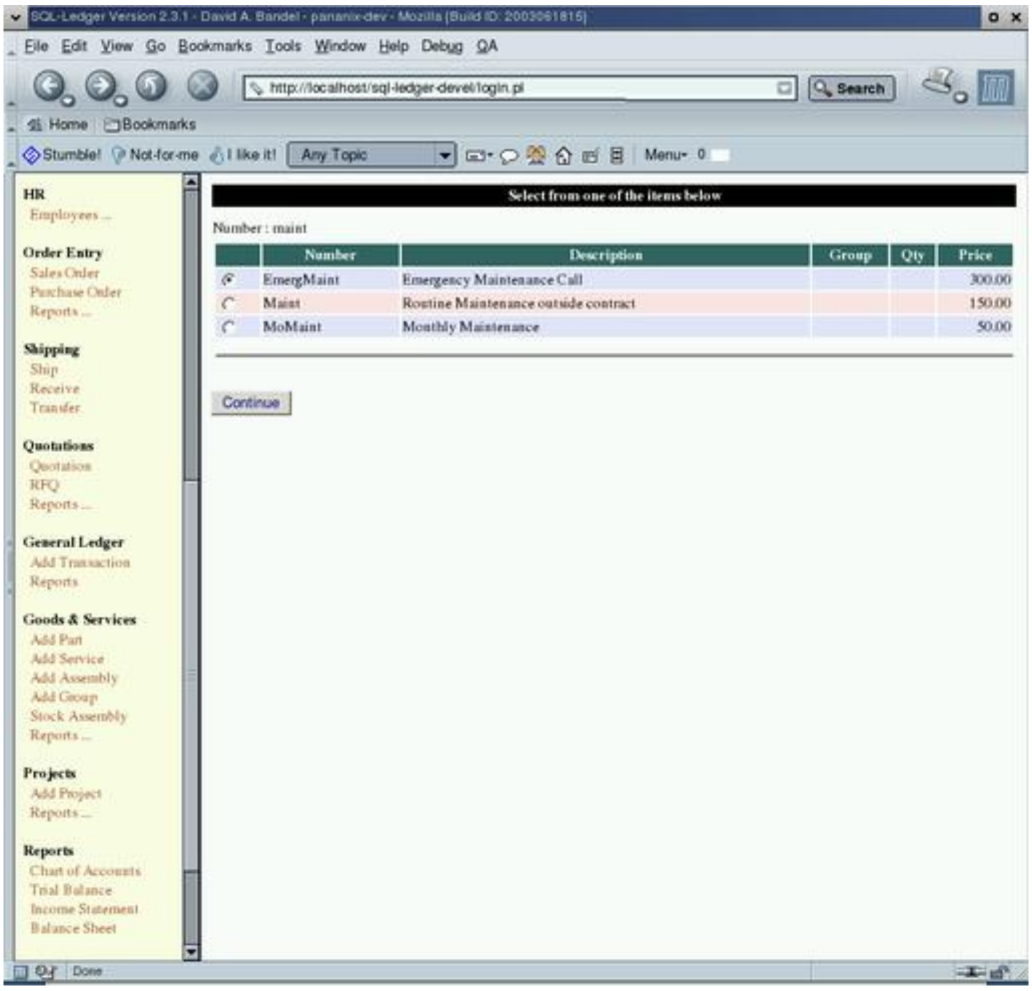


AR Transactions

Customer					
Invoice Number					
Order Number					
Notes					
From		To			
Include in Report	✓ Open	Closed			
	ID	✓ Invoice Number	Order Number	✓ Invoice Date	
	✓ Customer	Amount	Tax	✓ Total	
	Date Paid	✓ Paid	Due Date	Amount Due	
	Notes	Till	Salesperson	Shipping Point	Ship via
	Subtotal				

Figure 2. AR Transactions Query Screen

Notice the Customer window at the top of the browser window in Figure 2. In a number of screens, this can be either a drop-down list or a pick list in a separate window. Better yet, this presentation is configurable by each user. So users who like drop-down lists and don't mind that the list scrolls 16 pages off the bottom of the screen can put a large number in their pick-list preference. Those that want a smaller, more sane list, can get a drop-down list unless the number of available items for the drop-down list exceeds the limit. Then, they simply can put in a few letters of the name for which they're searching, refresh the screen and get a small pick list, as shown in Figure 3. This particular pick list came up from an invoice screen after entering `maint` in the part number window and selecting Update.



Select from one of the items below

Number : maint

	Number	Description	Group	Qty	Price
◉	EmergMaint	Emergency Maintenance Call			300.00
○	Maint	Routine Maintenance outside contract			150.00
○	MoMaint	Monthly Maintenance			50.00

Figure 3. Sample Pick-List Screen

For those of you interested in playing with SL without going to the trouble of setting it up, I suggest you head over to www.sql-ledger.com. A few demo systems are available, so folks can try before they install the software.

Using SQL-Ledger in Your Business

I'm not an accountant, and I barely get by with my accounting duties as it is. As with most powerful accounting software, you should know something about it before using it. But if it's an accounting question, I probably don't have the answer.

Fortunately, except for the few times I may have to wade into general ledger, the system takes care of itself. I've found the customer list to be quite friendly. It serves as both a customer list and a way to keep e-mail addresses straight. I e-mail monthly bills, and this program has made that particularly easy. In fact, after posting, you can call up a bill and e-mail it with a single click. The program handles it all for you, including sending a cover letter.

Purchase orders are as easy to do, and they also can be e-mailed directly from the interface. When orders come in it's a few more clicks to enter inventory and create the payable. A quick trip to Cash→Payments and we're done.

If you have several businesses, you can run as many ledgers as you want using different database names. You also have to use different user names, one for each, but I've found the easy way to do this is to make the user name a combined user/company name. You may think of something more convenient.

Now that the latest SL includes POS, you can connect a bar-code reader and use that to enter items on the screen. SL was designed deliberately with the UPDATE button first, so scanning a bar code fills in a product number then updates that to fill in the rest of the line.

Coming Attractions?

SL is now fairly feature-rich these days, especially compared to several years ago, but the author maintains an ambitious to-do list. By the time you read this, in fact, SL even may contain payroll. Most of the tables and links are in place; it appears to be a matter of coding and testing. I fully expect a payroll system to rival any available, based on what's come before in SL.

The SL to-do list includes such items as:

- Budgets: comparisons to actual ones and also to enforce budget (expense) constraints.
- Manufacturing: finished goods and goods-in-process inventory; material and human resources planning.
- Lot allocation: repackaging bulk goods into smaller portions; memorized transactions and custom reports.
- Batch form generation: invoices, orders and other forms for batch printing.
- Financial reports: more comparison options, such as month-to-month.

Those interested can take a look at SL's What's Ahead page.

Support

SL offers a range of support for the software. From the SL home page you can find several users lists in a variety of languages. Currently six user lists exist. The author lurks on the English list. He occasionally posts to clear up misunderstandings.

Paid support also is available at extremely reasonable rates. A wide variety of support options are available, and most come with a copy of the SQL-Ledger users manual. If you want to use this software without support, that's fine. But as with all GPL software, if you break it, you get to keep both pieces. Paid support is your assurance it doesn't break. A support option is available to import old data from accounting packages that can export tab-delimited text files.

Conclusion

Although changing accounting packages always is an ordeal, SL is one package worth investigating. The price can't be beat, and this software competes against and beats many proprietary offerings. And if you don't see it but need it, you always can request a feature.

David A. Bandel (david@pananix.com) is a Linux/UNIX/Network (both wired and wireless) consultant in Panama who dabbles in almost all aspects of telecommunications. He's authored or coauthored three Linux books, runs two HF radio Sailmail stations and does volunteer work for the Linux Professional Institute. When not working, he can be found relaxing on his farm near the Costa Rican border.

Automating Tasks with Aap

Bram Moolenaar

Issue #121, May 2004

Aap is a flexible tool that can do what **make** does and much more. Learn how to write portable recipes for maintaining your Web sites and building programs.

Many people use a Makefile and shell scripts to automate tasks where a change to a file requires an action to be taken. You edit a file and then invoke `make` in the hope that all actions necessary to effect the changes are done. Often you need to tweak the Makefile to get it right and end up using `touch` to work around a missing dependency. As a result, when other people look at your carefully tuned Makefile, they have a hard time understanding how it works.

These tasks can be done more easily and reliably with Aap than with make. For example, Aap has built-in Internet support. Downloading and uploading is taken care of without the need to specify the commands or to keep timestamp files. Reliability is achieved by figuring out dependencies automatically and using signatures instead of timestamps. With Aap it is simpler to specify the work you want done, and you make fewer mistakes. You still can fall back on using shell commands where you want them. This article presents two examples of Aap in action, maintaining a Web site and building a program. Aap can do much more, of course, but these subjects should be sufficient to get you started.

Installing Aap

To use Aap you need Python version 1.5 or later. In the unlikely event that you do not have Python on your system, download it from Python.org, or install it from your Linux distribution CD or update system.

Installing Aap can be done in four simple steps. Start by downloading the latest Aap zip archive (see the on-line Resources section). Then, unpack the archive in a temporary directory (`unzip aap-1.053.zip`). If you are root, run `./aap install`. If you are a normal user, install the archive in your home directory with `./aap install PREFIX=$HOME`. For more information about downloading and installing Aap, see Resources. Aap is distributed as open-source software under the GNU GPL.

Maintaining a Web Site

With the current Aap software in place, let's look at using it in a basic example task. You have designed a simple Web site with HTML files and images. The files are on your local computer, and you need to upload them to the Web server. Listing 1 shows the Aap script, called a recipe, that does this.

Listing 1. Recipe for Uploading Files to a Web Server

```
# The list of files to be uploaded.
Files = index.html
        info.html
        download.html
        images/*.png

# The publish attribute tells where to upload to.
:attr {publish = scp://my.server.net/html/%file%}
        $Files

# When executed without a target: publish the files.
all : publish
```

Store the recipe under the name main.aap. This file is what Makefile is to **make**—the default file to be executed. Running `aap` without arguments executes the main.aap recipe in the current directory.

Comments in an Aap recipe start with # and continue until the end of the line, as in a Makefile or shell script. The first effective line in the recipe is an assignment; the list of files to be uploaded is assigned to the Files variable. There are no back slashes nor punctuation to mark the end of the assignment. Aap recognizes command continuation by the amount of indentation used. This may appear strange at first, but you quickly get used to it. This method avoids the usual mistakes with punctuation and enforces a layout that is easy to read. Aap allows you to use either tabs or spaces to indent lines.

The :attr line is an Aap command. All Aap commands start with a colon to make them easy to recognize. This command adds the publish attribute to its arguments. The publish attribute tells Aap where to upload the files when they are published. The method used here is scp://, secure copy. Other supported methods are rsync:// and ftp://. The last argument of :attr is $Files, the value of the Files variable. The attribute is attached to each item in $Files.

When you run Aap without an argument, it updates the target all. The final line specifies that the default target all depends on publish. This is a special target, which tells Aap to upload all items that have a publish attribute.

You now can edit the HTML files, add pictures and view them locally. Once you are satisfied, execute `aap`. Aap figures out which files have changed and uploads them. Signatures (checksums) are used; thus, if you restore an old version of a file it still works properly. If you were using make, you would have to touch the restored file to update its timestamp.

If you want to try this example, but you don't have a server to upload to, you can use the publish attribute `file:/tmp/html/%file%`. With it, Aap creates the /tmp/html directory, if necessary.

A word of warning: Aap does not delete files on the server no longer in use. Then again, neither does make. You have to delete these files manually. Hopefully, automatic deletion will be added to Aap soon.

Listing the Image Files

A wild card was used to select the images: `images/*.png`. This is convenient, but it has the danger of including images you do not want uploaded. Explicitly naming each file avoids this trap, but then you might forget an image. Being that this is a common issue, Aap provides a function to extract the image filenames from the HTML files. Listing 2 shows how this is done; the Python function get_html_images is invoked, and the back ticks enclose a Python expression. Aap evaluates the expression and puts the result, the image filenames, in its place. The get_html_images() function has limited capabilities, however. It works only for plain HTML files with images that have a relative pathname.

Listing 2. Getting the Image Filenames from HTML Files

```
# The list of files to be uploaded.
Files = index.html
        info.html
        download.html
Files += `get_html_images(Files)`

# The publish attribute tells where to upload to.
:attr {publish = scp://my.server.net/html/%file%}
        $Files

# When executed without a target: publish the files.
all : publish
```

Generating HTML Files

Most HTML files consist of a header, title, main contents and footer. Obviously, you don't want to type the common parts each time. A simple solution is to concatenate a number of files. Listing 3 shows the recipe that implements this. Five parts are used: header, title, middle, contents and footer. The title and contents are different for each page, but the other three parts are the same.

Listing 3. Generating an HTML File from Five Parts

```
Files = index.html
        info.html
        download.html

:rule %.html : header.part
               %_title.part
               middle.part
               %.part
               footer.part
    :cat $source >! $target

:update $Files
Files += `get_html_images(Files)`

:attr {publish = scp://my.server.net/html/%file%}
        $Files

all : publish
```

The main difference between Listing 2 and Listing 3 is the added :rule command in Listing 3. It specifies that a target (the HTML file) depends on five source files (the parts) and lists the command to build the target from the sources. The % character is used instead of the name of the file, similar to a * wild card. All % characters in the rule stand for the same name. Thus, for index.html the % stands for index. The sources then include index_title.part and index.part.

Below the :rule line comes the indented block of statements that are executed when the target of the rule needs to be updated. So, the recipe has two levels: the commands at the top level are executed when reading the recipe, and the command block of the rule is executed later, when needed.

The :cat command concatenates files, the same as the UNIX cat command. It actually can do much more, such as read files from a specified URL. In a rule, $source stands for the whole list of source files.

The HTML files need to be generated before obtaining the list of image files they contain. To get this right, the :update command is invoked before calling get_html_images(). The HTML files are updated using the defined rule. This is at the top level of the recipe, so it always is done when Aap reads the recipe.

Now that you have so many files, how does Aap keep track of what needs to be done? Aap works with dependencies, the same as make does. It starts with the target you specify on the command line. When no target is given, all is assumed. Aap then locates those dependencies and rules in which this target appears before the colon. The colon basically means depends on; after the colon are the source files on which the target depends. Each of these source files then is inspected, and Aap finds rules where they appear as a target. This continues recursively until no more rules are found. The result is a tree of dependencies. Aap then executes commands for those dependencies that need to be built, starting at the end of the tree (depth first). This sounds complicated, doesn't it? Because Aap takes care of this, you only need to make sure you specify the sources on which each target depends. Aap figures out what needs to be done.

Adding a Timestamp

As a nice addition, let's add a timestamp to the HTML file, so you can see on the Web site when the page was last generated. Put the string @TIMESTAMP@ somewhere in the file footer.part. Listing 4 shows the rule in which this string is replaced with the current date. The rest of the recipe is as shown in Listing 3. The :eval command evaluates a Python expression, and string.replace is a standard Python function for replacing one string with another. This way, you can use any Python expression to filter text. The HTML page is piped through the :eval command, as with a shell.

Listing 4. Rule for Putting a Timestamp in a Generated HTML File

```
:rule %.html : header.part
               %_title.part
               middle.part
               %.part
               footer.part
   :print Generating $-target
   :cat $source
       | :eval string.replace(stdin,
                         '@TIMESTAMP@', _no.DATESTR)
       >! $target
```

The first time the new rule is used, all HTML files are updated. That is because Aap remembers a signature for the commands. Thus, you don't need to worry about forcefully generating the files after changing the commands in the recipe.

Uploading with rsync

When making small changes to a Web page, it is a waste of bandwidth to upload the whole file each time. A good way to upload efficiently is to use rsync. It uploads only those parts of a file that have been changed. Aap uses rsync when it finds rsync:// in the publish attribute. By default, rsync is used over an SSH connection. You can change this by setting the $RSYNC variable.

rsync is not a standard command. If it does not exist on the system, you encounter a nice feature of Aap—you are offered the choice to install rsync:

```
% aap
Aap: Uploading ['index.html'] to
               rsync://my.server.net/html/index.html
Cannot find package "rsync"!
1. Let Aap attempt installing the package
2. Retry (install it yourself first)
q. Quit
Choice:
```

Aap has a mechanism to install a package when it is needed by downloading a recipe from the Aap Web site that specifies how the package is to be installed. The downloading features of Aap come in handy here. How the package is installed depends on your system; not all systems are supported yet. After rsync has been installed, Aap starts uploading the files.

Building a Program

Aap includes support for building a program from C and C++ code. Here is the one-line recipe that builds the program called myprog from four C source files:

```
   :program myprog : main.c common.c various.c args.c
```

Despite the simplicity of the recipe, Aap takes care of several issues:

- Dependencies are figured out automatically. You don't need to specify the included header files or do a `make depend`.
- This recipe works on most systems without modification. Aap finds a compiler and linker to use and figures out the arguments they need.
- The object files are stored in a separate build directory for each system. You can build several versions without cleaning up.
- Aap creates a log file, AAPDIR/log, that contains details about what happened. If your build fails and the output scrolls off the screen, you don't need to repeat the build command with the output redirected.
- A few default targets are added automatically: `aap install` installs the program, and `aap clean` deletes generated files.

It would be possible to do the same work with make, with the help of a few extra tools. But the Makefile would be much longer and not portable; it also would require more effort to maintain.

Building Variants

Now let's build a program in two variants, a release and a debug version. Aap includes support for variants. All you need to do is specify what variants you want to build and what is different between them. Listing 5 shows the recipe.

Listing 5. Building Release and Debug Variants

```
   :variant Build
      release
         OPTIMIZE = 4
         Target = myprog
      debug
         DEBUG = yes
         Target = myprogd

   :program $Target : main.c common.c various.c args.c
```

The first line of the :variant command specifies the variable name used to select the variant to be built. You can set this variable on the command line; `aap Build=debug` builds the debug version. Without an argument, the release variant is built, because it is mentioned first.

The amount of indentation identifies the other parts of the :variant command. The possible values have less indentation; the commands used for each value have a bit more. You are forced to align the parts, which makes them easier to read.

The release variant sets the OPTIMIZE variable. This is a number in the range of zero to nine that indicates the amount of optimizing to be done. It automatically is turned into the right argument for the compiler being used. The debug variant sets DEBUG to yes. The default value is no. The Target variable holds the name of the resulting program. The two variants use a different name, so both programs can exist.

A nice advantage of using variants this way is that object files for each variant are stored automatically in a separate build directory. When switching between the two variants you should notice that Aap does not rebuild all the files.

Building with Another Language

For languages other than C and C++ you need to import a language module. A few standard modules are included with Aap. For example, this is how to build from D sources; D is a new programming language:

```
:import d
:program myprog : main.d common.d various.d args.d
```

The :import d command is used to load the support for the D language. Otherwise this process is similar to building from C sources.

You can write a module yourself to add support for a language. Because Aap is open source, you are encouraged to submit the module to be included in the Aap distribution. Until that happens, drop the file in the Aap modules directory; this works as a plugin.

Building a KDE Application

Building a KDE application involves working with a lot of tools, including using Qt Designer to create dialogs, generating header files from user-interface descriptions and generating interprocess communication code. Nevertheless, a recipe for building a KDE application can be as simple as this:

```
:import kde
:program logger : main.cpp
                  logwidget.ui
                  dcop.h {filetype = skel}
                         {var_OBJSUF = _skel.o}
```

Of the three input files, main.cpp can be compiled directly. The Qt Designer file logwidget.ui first needs to be processed by uic to generate an include file; then moc must be used. Aap recognizes the .ui suffix and takes care of all of this. Handling this kind of multistep compilation, from ui to h to moc to object file, is a useful feature in Aap. Doing the same thing in a Makefile requires far more explicit rules.

The dcop.h file contains special KDE items but has a normal suffix. It cannot be recognized automatically. Therefore the filetype attribute is specified explicitly. The :program command also needs to know the name of the object file, which is specified with the var_OBJSUF attribute. You do not need to specify explicitly the KDE tools being used; the complexity is hidden in the KDE module. This is considerably less complex than using automake.

Using Aap as a Better make

So far, you have used high-level Aap commands to specify quickly what needs to be done. For nonstandard tasks, you need to spell out the dependencies and commands. This mostly works like a Makefile. Besides shell commands, you can use portable Aap commands. If that is not enough, you can add a Python script.

Listing 6 shows what a low-level recipe looks like. Every dependency is given explicitly here—all depends on hello, hello is compiled from hello.c and hello.c is generated from scratch.

Listing 6. Using Aap as a make Replacement

```
all : hello

# Manually compile the hello program.
hello : hello.c
   :sys cc -o $target $source

# Clumsy way to generate a C program.
hello.c:
   :print Generating $target
   :print >! $target $(#)include $(<)stdio.h$(>)
   :print >> $target main() {
   :print >> $target    printf("Hello World!\n");
   :print >> $target    return 0;
   :print >> $target }
```

Because the build commands in a recipe are Aap commands, you need to use :sys to execute a shell (system) command. In the example, `:sys cc` executes the C compiler. Obviously, this works only on systems with the cc command. Using shell commands reduces the portability of a recipe.

The hello.c file is generated with :print commands. The first line uses `>! $target` to overwrite an existing hello.c file. Without the exclamation mark, you receive an error message if the file already exists. This line also contains $(#), which escapes the special meaning of the # character to start a comment. Likewise, $(<) and $(>) are used to get < and > characters instead of redirection.

The hello.c file is generated when it doesn't already exist; no source file dependency is specified. The file can be generated in another situation as well—if you change one of the :print commands, because it changes the signature of the build commands. When the build commands change, Aap knows that the target must be rebuilt.

The file is generated with Aap commands; no shell commands need to be used. This part of the recipe therefore can work on any system. But the number of Aap commands is limited. When you need more functionality and also require portability, you can use Python scripting.

All flow control in Aap recipes is done with Python, and Listing 7 illustrates an example of a recipe that applies patches to Vim. A loop is used to generate a list of patch filenames, starting with vim-6.2.001 and counting up to the last patch number, specified with LASTPATCH. Each of the patch files is to be downloaded and applied. The $* in done/$*Patches is used for rc-style variable expansion; done/ is prepended to every item in Patches.

Listing 7. Using Python to Create a List of Names

```
LASTPATCH = 144

# Generate a list of patch filenames.
@Patches = ''
@for i in range(1, int(LASTPATCH) + 1):
@   Patches = Patches + ("6.2.%03d " % i)

# Default target: apply all patches.
all: done/$*Patches

# Make sure the two directories exist.
:mkdir {force} patches done

# Rule for applying a patch.
:rule done/% : patches/% {fetch =
                 ftp://ftp.vim.org/pub/vim/%file%}
   :sys patch < $source
   :touch $target
```

Normally, you don't need to use much Python in your recipe, but it is good to know that complicated tasks are possible to accomplish when they arise.

Installing Packages

We already mentioned that Aap can install rsync for you if it cannot be found on your system. The package install mechanism also can be invoked directly. For example, to install Agide use the command `aap --install agide`. Agide is the A-A-P GUI IDE, another part of the A-A-P Project. You can use it to build and debug programs with Vim and gdb. It still is in an early stage of development, but it is good enough to develop and debug C programs.

Several packages currently are available, and more will be added over time. A list of the current packages can be found on www.a-a-p.org/packages.html. Aap itself also can be installed. Updating to the latest version can be done with `aap --install aap`. This command overwrites any existing Aap version. If your system has a package manager, you probably should use that instead.

Conclusion

You now have an idea of the tasks that you can automate with Aap. When you start experimenting you can find a lot of help in the comprehensive documentation. You can find it on the Aap Web site in several forms (see Resources). These pages explain many things that could not be included in this article, such as using CVS for version control, automatic configuration and so on.

Resources for this article: /article/7458.

Bram Moolenaar is the project leader and main author of Aap. He is known mostly for his work on Vim, the text editor. Bram's work on Aap was funded by Stichting NLnet www.NLnet.nl. You can find his home page at www.Moolenaar.net.

How to Build LSB Applications

Stuart R. Anderson

Issue #121, May 2004

Don't leave your Linux software stuck on one distribution. Make it run anywhere with the standard that all the major distributions use.

The Linux Standard Base (LSB) specifies an interface between an application and a runtime environment. Many distributions have achieved certification for their runtime environments. This article outlines the steps needed to build applications that adhere to the LSB interface.

Origins of the LSB

The LSB Project was founded in 1997 to address the application compatibility problem that was beginning to emerge. Different distributions were using different versions of upstream software and building them with different options enabled. The result was that an application built on one distribution might not run on another distribution. Worse yet, the application often would not work on a different version of the same distribution.

Originally, the LSB was intended to create a common reference implementation for the base of a GNU/Linux system. In addition to the reference implementation, a written specification was to be developed. This idea wasn't well received by many of the distributions that had considerable investments in their own base software, which they perceived as being a competitive advantage.

After further discussion among the interested parties, the LSB Project underwent a fundamental shift in focus in order to achieve consensus among the entire community. The shift gave priority to the written specification over the implementation, and it defined the LSB as a behavioral specification instead of a list of upstream feature/version pairs. This new focus was realized as a three-prong approach: a written specification, which defines the behavior of the system; a formal test suite, which measures an implementation against the specification; and a sample implementation, which provides an example of the specification.

Structure of the LSB

The LSB Specification actually is made up of a generic portion, the gLSB, and an architecture-specific portion, archLSB. The gLSB contains everything that is common across all architectures; we try hard to define as much as possible in the gLSB. The archLSBs contain the things that are unique to each processor architecture, such as the machine instruction set and C library symbol versions.

Contents of the LSB

As much as possible, the LSB builds on existing standards, including the Single UNIX Specification (SUS), which has evolved from POSIX, the System V Interface Definition (SVID) and the System V Application Binary Interface (ABI). The LSB uses the ELF definitions from the ABI and the interface behaviors from the SUS. It adds the formal listing of what interfaces are available in which library as well as the data structures and constants associated with them. See the “Linux Standard Base Libraries” sidebar for the list of libraries currently specified.

Linux Standard Base Libraries

As of LSB 1.3, the following shared libraries are specified in the LSB. All other libraries must be linked statically into the application.

Base libraries: libc, libm, libpthread, libpam, libutil, libdl, libcrypt, libncurses and libz.

Graphics libraries: libX11, libXt, libXext, libSM, libICE and libGL.

As the LSB continues to grow in future versions, so will this list of libraries.

In addition to the ABI portion of the LSB, the specification also specifies a set of commands that may be used in scripts associated with the application. It also mandates that the application adhere to the filesystem hierarchy standard (FHS).

One additional component of the LSB is the packaging format. The LSB specifies the package file format to be a subset of the RPM file format. The LSB does not specify that the distribution has to be based on RPM, however, only that it has some way of correctly processing a file in the RPM format.

One final item to mention is the name of the program interpreter. The program interpreter is the first thing executed when an application is started, and it is responsible for loading the rest of the program and shared libraries into the process address space. Traditionally, /lib/ld-linux.so.2 has been used, but the LSB specifies /lib/ld-lsb.so.1 instead on IA32. Generally, /lib/ld-*arch*-lsb.so.1 is used for other architectures. This provides the operating system with a hook early in the process execution in case something special needs to be done to provide the correct runtime environment to the application. You can pass the following to GCC to change the program interpreter:

```
  -Wl,--dynamic-linker=/lib/ld-lsb.so.1
```

The tools described here take care of this process for you.

The LSB Build Environment

A long time ago, people realized that code changes are cheaper and easier to make when they come earlier in a development process rather than later. With this in mind, the LSB Project has created a build environment to assist with the creation of LSB-conforming applications. This build environment provides a set of clean headers, stub libraries and a compiler wrapper.

The LSB stores much of its definition in a database. In addition to the portions of the specification that would be tedious to edit manually, we are able to produce a set of clean header files and stub libraries that contain only the things specified by the LSB. Using the database in this way helps to ensure the tools and specification stay in sync as changes and additions are made. The packages you need to install are described in the “Linux Standard Base Packages” sidebar.

Linux Standard Base Packages

You can get the LSB development environment from the Linux Standard Base (see the on-line Resources section); simply follow the links for downloads. You should install the following packages:

- lsbdev-base: contains the headers and libraries.
- lsbdev-cc: contains the compiler wrapper tools.
- lsbdev-chroot: contains the alternate chroot-based environment.
- lsbdev-c++: contains a static libstdc++, which can be used to port some C++ applications for LSB 1.3.

The first step in building an LSB-conforming application is to compile the code with the LSB headers. If the code doesn't compile, it probably is using something outside of the LSB. This isn't necessarily a showstopper, but it is something to which you need to pay particular attention. The LSB headers are installed in /opt/lsbdev-base/include. As a quick test, pass `-I/opt/lsbdev-base/include` to GCC and see what happens. The compiler wrapper described later does this step and some other related steps for you.

Once you have compiled your code, the next step and next test is to link the code together to form the final application. Usually, this step looks like this:

```
gcc -o app1 obj1.o obj2.o -lfoo
```

The LSB stub libraries can be found in /opt/lsbdev-base/lib and can be specified by passing the -L option to the compiler. These stub libraries are used only at link time. Typically, the normal system libraries are used at runtime. Again, the compiler wrapper described later handles these details for you.

Once you have linked your application, use the `ldd` command to see what shared libraries are being used. At this point, there should not be any shared libraries other than the ones specified in the LSB (and listed in the "Linux Standard Base Libraries" sidebar). If there are, you need to take extra steps to make them be linked statically. Usually, the `-Wl,-Bstatic` and `-Wl,-Bdynamic` options can be used to specify that certain libraries should be linked statically. By now, you may be seeing a pattern: the compiler wrapper handles this for you.

As an example, here is what the application xpdf typically looks like:

```
# ldd /usr/bin/xpdf
  libXpm.so.4 => /usr/X11R6/lib/libXpm.so.4
  libt1.so.1 => /usr/lib/libt1.so.1
  libfreetype.so.6 => /usr/lib/libfreetype.so.6
  libSM.so.6 => /usr/X11R6/lib/libSM.so.6
  libICE.so.6 => /usr/X11R6/lib/libICE.so.6
  libX11.so.6 => /usr/X11R6/lib/libX11.so.6
  libpaper.so.1 => /usr/lib/libpaper.so.1
  libstdc++-libc6.2-2.so.3 =>
         /usr/lib/libstdc++-libc6.2-2.so.3
  libm.so.6 => /lib/libm.so.6
  libc.so.6 => /lib/libc.so.6
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2
```

Here is the LSB-conforming xpdf:

```
# ldd /opt/lsb-xpdf/bin/xpdf
    libSM.so.6 => /usr/X11R6/lib/libSM.so.6
    libICE.so.6 => /usr/X11R6/lib/libICE.so.6
    libX11.so.6 => /usr/X11R6/lib/libX11.so.6
    libm.so.6 => /lib/libm.so.6
    libgcc_s.so.1 => /lib/libgcc_s.so.1
    libc.so.6 => /lib/libc.so.6
    /lib/ld-lsb.so.1 => /lib/ld-lsb.so.1
```

The non-LSB libraries are not showing up as needed by the application, because they are linked statically into the application itself. There is a trade-off here: the application executable becomes larger, but it has fewer dependencies on the installed operating system.

Making It Easy

Finally, we get to the compiler wrapper, lsbcc and lsbc++. These are the same program; they simply are invoked with different names to indicate C or C++ mode. The general idea is you can use lsbcc wherever you would use GCC and lsbc++ wherever you would use g++.

This wrapper tool parses all of the options passed to it and rearranges them slightly. It then inserts a few extra options to cause the LSB-supplied headers and libraries to be used ahead of the normal system libraries. This tool also recognizes non-LSB libraries and forces them to be linked statically.

Because the LSB-supplied headers and libraries are inserted into the head of the search paths, it generally is safe to use things not in the LSB. Make sure, however, that they are not dependent on something that intentionally has been left out of the LSB headers and libraries and that they can be linked statically into the applications. This allows lsbcc to be transparent in most cases.

Using the LSB Development Environment

With the LSB development packages installed, porting a sample application becomes as easy as the normal three-step process, but with a slight difference:

```
CC=lsbcc ./configure
make
make install
```

By telling the configure script to use lsbcc instead of GCC, it conducts its various tests in an LSB environment and configures the software with any adjustments or limitations that may be required. Sometimes this results in a portable replacement for a feature being used. Generally, though, the overall functionality is close to what it would have been if GCC were used instead. As an exercise, try running a configure script both ways and compare the results. Another benefit of telling configure to use lsbcc is that it automatically sets CC to lsbcc in the generated makefiles, so you don't have to remember to pass it in (`make CC=lsbcc`) every time you run `make`.

The lsbcc command defaults to calling GCC with the modified arguments, but an environment variable can be used to tell it what compiler to use instead. This should work okay for any other compiler that is command-line option compatible with GCC.

Testing Tool

Once the application has been built, use the lsbappchk program to test the program to see if it conforms to the LSB. This program checks the list of shared libraries used by your application; it also checks to make sure you are using only the interfaces permitted by the LSB. Here is an example run:

```
# /opt/lsbappchk/bin/lsbappchk /bin/ls
/opt/lsbappchk/bin/lsbappchk for LSB Specification 1.3.3
Checking binary /bin/ls
Incorrect program interpreter: /lib/ld-linux.so.2
Header[ 1] PT_INTERP    Failed
Found wrong interpreter in .interp section: /lib/ld-linux.so.2
                               instead of: /lib/ld-lsb.so.1
DT_NEEDED: librt.so.1 is used, but not part of the LSB
Symbol clock_gettime used, but not part of LSB
```

The LSB does not require that the utilities provided by the OS be LSB-conforming themselves. Therefore, there isn't really an expectation that a distribution's own /bin/ls should pass this test. It simply makes for a handy example.

The output of lsbappchk tells us that /bin/ls is not an LSB-conforming application. The first problem is it wasn't linked with the LSB-defined program interpreter /lib/ld-lsb.so.1. The next problem is that the application is looking for the shared library librt.so.1, which is not included in the set of LSB-defined libraries. Lastly, the function clock_gettime() is used but is not linked statically to the application (it would have been found in librt.so.1).

The general approach to fixing an application such as this would be to rebuild the application using lsbcc, which would set the program interpreter correctly and cause librt.a to be used instead of librt.so. Sometimes, statically linking a library can cause new non-LSB symbols to be brought into the application, so this process may have to be repeated a couple of times.

In some larger applications or in sets of related applications, it may be desirable to create shared libraries that are used only by these applications. This is permissible under the LSB as long as the shared library is installed as part of the application and it resides in the application private data area, not in any of the system library locations. The -L option to lsbappchk lets you tell the testing tool the full path to the shared library, which is considered to be a part of the application for the purpose of testing conformance to the LSB. Here is an example of an LSB-conforming build of the Apache Web server, which uses three private shared libraries:

```
# /opt/lsbappchk/bin/lsbappchk \
  -L /opt/lsb-apache/lib/libaprutil.so.0 \
  -L /opt/lsb-apache/lib/libexpat.so.0 \
  -L /opt/lsb-apache/lib/libapr.so.0 \
  /opt/lsb-apache/sbin/httpd
```

```
/opt/lsbappchk/bin/lsbappchk for LSB Specification 1.3.3
Adding symbols for library /opt/lsb-apache/lib/libaprutil.so.0
Adding symbols for library /opt/lsb-apache/lib/libexpat.so.0
Adding symbols for library /opt/lsb-apache/lib/libapr.so.0
Checking binary /opt/lsb-apache/sbin/httpd
```

Packaging

As I mentioned earlier, the LSB specifies that a package must be delivered in the RPM file format. This does not mean that RPM has to be used to build or package your application, although it may be the most practical option, depending on whether you already are using it. Other options would be creating the package in the Debian format, and then using alien to convert it to RPM. Or, you could use some other tool for creating the RPM file format. We have the beginnings of a tool called mkpkg to create the RPM format file, but it likely will require something to sit on top of it to make it useful to any but the most die-hard hacker.

In our application battery, we currently build the application and install it in a temporary root and then invoke RPM to package up the install application. This may seem a little clunky, but it works without much pain and produces more consistent results across all of the different versions of RPM found in the wild.

Here is a sample spec file for the xpaint application:

```
Summary: An X Window System paint program
Summary: XPaint
Name: lsb-xpaint
Version: 2.6.2
Release: 3
Vendor: Free Standards Group
License: MIT
Group: Appbat/graphics
Buildroot: /usr/src/appbat/pkgroot/lsb-xpaint
AutoReqProv: no
PreReq: lsb >= 1.3

%description
LSB conforming version of xpaint. XPaint is an
X Window System color image editing program and
painting program. Xpaint is added to the LSB
Application Battery primarily to demonstrate the
use of X11 libraries.

%pre

%install

%post

%preun

%postun

%clean

%files

%attr ( - bin bin ) /opt/lsb-xpaint
```

Full source code for building and packaging this and the other applications in the application battery can be found in the LSB Project CVS tree.

Does This Really Work?

Yes, it really does work, although to be fair, we still are running into corner cases and various applications that don't always follow the rules for clean, portable code. As part of the verification for the LSB, we have created an application battery built from the tools described here. This set of applications includes Apache, Samba, Lynx, Python, xpdf and groff. We have tried to select a set of real applications that provide coverage over as much of the LSB set of interfaces as possible.

What about C++ Applications?

LSB version 1.3 does not support C++, so the rule requiring the library to be linked statically applies. We are adding support for C++ to LSB 2.0 to avoid this. We provide the lsbdev-c++ package, which contains a version of libstdc++ that was configured and built with lsbcc. This and GCC version 3.2 seem to produce good results. We have tried other combinations of compilers and different versions of the C and C++ libraries but ran into various problems, depending on the nature of the application.

Future Directions

For the LSB in general, we will continue to add additional libraries to the specification as long as there is consensus that they are needed and have reached a certain level of stability. This should help close the gap between how distribution-provided and LSB-conforming applications are built.

For the LSB development environment, we will continue to make the tools better and more transparent. The development environment is being maintained actively, and feedback from people using these tools is appreciated. With the addition of C++ in LSB 2.0, the development environment will be able to drop the lsbdev-c++ package being used today in favor of the C++ stub library, which will move into the base LSB development package.

Currently, you may have to set several options in an rpmrc or rpmmacros file to make RPM produce LSB-conforming packages. It is our hope that we can come up with an LSB mode for rpmbuild that can handle all of this automatically. Hopefully, it will make it even easier to build existing packages that conform to the LSB.

Acknowledgements

First off, thanks to the Free Standards Group and its members for providing the support to the LSB Project that has enabled us to accomplish as much as we have. Secondly, thanks to the core group of developers working on the development environment for the LSB, including Chris Yeoh, Marvin Heffler and especially Mats Wichmann for their patience and persistence during the more experimental phases of this project.

Resources for this article: /article/7459.

Stuart R. Anderson (anderson@freestandards.org) made the mistake of being overheard while saying “I know how to fix that”, and he has been the lead developer of the LSB Written Specification ever since. When not working on the LSB, Stuart keeps busy enlightening South Carolina to open-source ideas by converting companies one at a time.

Archive Index Issue Table of Contents

Advanced search

Shielded CPUs: Real-Time Performance in Standard Linux

Steve Brosky

Issue #121, May 2004

Reserve one processor for a high-priority task and improve real-time performance.

In a multiprocessor system, a shielded CPU is a CPU dedicated to the activities associated with high-priority real-time tasks. Marking a CPU as shielded allows CPU resources to be reserved for high-priority tasks. The execution environment of a shielded CPU provides the predictability required for supporting real-time applications. In other words, a shielded CPU makes it possible to guarantee rapid response to external interrupts and to provide a more deterministic environment for executing real-time tasks.

In the past, a shielded CPU could be created only on symmetric multiprocessing systems. With the advent of hyperthreading (where a single CPU chip has more than one logical CPU), even a uniprocessor can be configured to have a shielded CPU.

The shielded CPU approach to providing high-end real-time performance allows the developer of a real-time application to achieve results comparable to the results achieved using a small real-time executive. For example, the results compare to approaches such as RTAI or RT/Linux, where Linux is run as one process under a real-time executive. The advantages of using a pure Linux environment for application development as opposed to one of these executives are many. For example, Linux has support for many device drivers, lowering the overall cost of implementing a complete application solution. A wide variety of high-level languages for better programming efficiency is supported. This is important for commercial applications; programming efficiency may not be central to the design of the real-time system, but it is helpful during the development phase and can provide additional functionality in the end system. Furthermore, Linux offers complex protocol stacks such as CORBA, extensive graphics capabilities and advanced application development tools.

Besides all of the functionality available in standard Linux today, an ever-expanding list of features is being developed for the Linux operating system, due to the strong momentum of the Linux phenomenon. By using Linux as the basis for an application design, a user will have many more options in the future.

Real-Time Means Guarantees, Not Merely Speed

A real-time application is one that must respond to a real-world event and complete some processing task by a given deadline. A correct answer delivered after the deadline becomes an incorrect answer. The deadlines themselves are application-dependent and can vary from tens of microseconds up to several seconds. For hard real-time applications, no deadlines can be missed. This means that worst-case measurements of system metrics are the only thing that matters to a hard real-time application, because these are the cases that cause a missed deadline.

Because the occurrence of a real-world event is communicated to a computer system by way of an interrupt, a real-time operating system must provide guaranteed worst-case interrupt response time. In responding to an interrupt and giving control to the real-time application, the computer system has performed the first step needed to meet the deadline. Once the real-time application is running, the system also must provide the application with deterministic execution times. If the time it takes to execute the code associated with a real-time application's response varies widely, deadlines are missed.

To guarantee good interrupt response, the operating system must be able to preempt quickly any tasks currently executing when an interrupt occurs. Because the 2.4 Linux series does not allow one task to preempt the execution of another task executing inside the kernel, a kernel based on this series has poor worst-case interrupt response. A preemption patch is available to make a task executing within the kernel preemptible. Even in a Linux kernel that has the preemption patch installed, however, a hidden problem exists that still causes long interrupt response delays.

The job of any operating system is to coordinate the execution of the many tasks sharing the resources of the system. The data structures that describe these shared resources can be corrupted if they are accessed by multiple tasks at the same time. Therefore, all operating systems have critical sections of code that can be accessed only by tasks in a sequential fashion. When a high-priority task suddenly becomes runnable—because an interrupt occurred—that task cannot take control of the CPU if another task currently is executing inside of one of these critical sections. This means that long, critical sections have a big impact on the ability of the system to respond to an interrupt. The low-latency patches address some of the longer critical sections in the Linux kernel by making algorithmic changes that shorten the critical sections.

In general, the more complex a subsystem is, the longer the critical sections. Because Linux supports many such complex subsystems, including the filesystems and networking and graphics subsystems, its critical sections are very long compared to the critical sections in a small real-time OS. The preemption patch and the low-latency patches have improved the responsiveness of Linux greatly. Still, many critical sections can last tens of milliseconds—not acceptable for the deadlines required by many real-time applications.

What Is a Shielded CPU?

As defined previously, a shielded CPU is dedicated to running a high-priority task and the interrupt(s) associated with that task. To create a shielded CPU, the operating system must provide the ability to set a CPU affinity for both processes and interrupts. The 2.4 series of Linux has the ability to set CPU affinity for interrupts, and open-source patches are available that provide this capability for processes. (See “Kernel Korner: CPU Affinity”, *LJ*, July 2003).

Because a shielded CPU does not run background tasks, a high-priority task on a shielded CPU never is prevented from responding to an interrupt because another task currently is executing inside of a critical section on that CPU. Interrupts always execute at a priority higher than any task, and because they occur at unpredictable points in time, non-real-time interrupts can cause significant non-determinism in a process' predicted execution time. A shielded CPU is not permitted to run interrupts unless the interrupt is one that a high-priority task on the shielded CPU is using.

Implementing Shielded CPUs

With the ability to set CPU affinity on processes and interrupts, it would be possible to set up a cheap implementation of CPU shielding. However, this implementation would rely upon all processes to honor the shielded CPU by not changing their affinity to include the shielded CPU. A stronger implementation is desirable, and one such implementation is described below.

The user interface for specifying CPU shielding is a /proc interface that allows an administrator to specify a mask of CPUs that are shielded, as well as a command that manipulates this mask. This interface allows a CPU to be marked dynamically as shielded. Once a CPU is shielded, no process can have its CPU affinity set to include the shielded CPU unless this prohibition precludes the process from executing on any CPUs. Thus, users must select a shielded

CPU specifically as the CPU where their tasks should execute in order to run on the shielded CPU. Only a privileged process can add CPUs to its affinity mask.

This implementation requires changes to the code that sets a process' affinity. The routine sys_sched_setaffinity() sets a CPU affinity. This routine is changed to remove a shielded CPU from any user-specified mask when a CPU affinity is set:

```
p->cpus_allowed_user = new_mask;
if (new_mask & ~shielded_proc)
    new_mask &= ~shielded_procs;
set_cpus_allowed(p, new_mask);
```

Notice that the shielded CPU bits are not removed if their removal would leave the process with no CPUs on which to execute. The field cpus_allowed_user is a new field in the task structure that holds the original process affinity as specified by the user. Whenever the mask of shielded CPUs changes, the code above needs to be reiterated over all processes in the system. This requires knowing the original CPU affinity for this process, as set by the user. The code that implements a change to the shielded CPU mask looks like this:

```
for_each_task(p) {
   new_mask = p->cpus_allowed_user & cpu_online_map;
   if (new_mask & ~shielded_proc)
      new_mask &= ~shielded_procs;
   if (new_mask != p->cpus_allowed)
      set_cpus_allowed(p, new_mask);
}
```

Performance Tests

To measure interrupt response time, the realfeel benchmark from Andrew Morton's Web site was used. This test was chosen because it uses the Real Time Clock (RTC) driver, a mechanism for generating interrupts common to many Linux variants. This test measures the response to an interrupt generated by the RTC driver. The RTC driver is set up to generate periodic interrupts at a rate of 2,048Hz. The RTC driver supports a read system call that returns to the user when the next interrupt has fired. The clock used to measure interrupt response is the IA-32 TSC timer, which has a resolution based on the CPU's clock speed. To measure interrupt response time, the test first reads the value of the TSC and then loops doing reads of /dev/rtc. After each read completes, the test finds the current value of the TSC. The difference between two consecutive TSC values measures the duration that the process was blocked waiting for an RTC interrupt. The expected duration is 1/2,048 of a second. Any time beyond the expected duration is considered latency in responding to an interrupt.

To measure worst-case interrupt response time, a strenuous background workload must be run on the system. This workload must provide the system with sufficient overhead to cause delays in the ability of the system to respond to interrupts as well as the resource contention that causes non-deterministic execution. The Red Hat stress-kernel RPM was chosen as the workload. The following programs from stress-kernel were used: TTCP, FIFOS_MMAP, P3_FPU, FS and CRASHME.

The TTCP program sends and receives large data sets over the loopback device. FIFOS_MMAP is a combination test that alternates sending data between two processes by way of a FIFO and operations on an mmaped file. The P3_FPU test manipulates floating-point matrices through various operations. The FS test performs all sorts of operations on a set of files, such as creating large files with holes in the middle, then truncating and extending those files. Finally, the CRASHME test generates buffers of random data, then jumps to that data and tries to execute it. Although no Ethernet activity is generated on the system, the system remains connected to a network and handles standard broadcast traffic during the test runs.

A new version of stress-kernel's NFS_COMPILE test was used because the original version had errors in its cleanup that prevented the test from being run for an extended period of time. The NFS_COMPILE script is the repeated compilation of a Linux kernel, using an NFS filesystem exported over the loopback device. The system used to run all tests was a dual-processor Pentium 4 Xeon with 1GB of memory and a SCSI disk drive.

Testing Results

RedHawk Linux version 1.3, from Concurrent Computer Corporation, was used to measure interrupt response on a shielded CPU. RedHawk is a Linux kernel based on kernel.org 2.4.21. It should be noted that shielded CPUs are only one of the real-time enhancements made to the RedHawk Linux kernel. Some of the other enhancements also contributed to the reported performance numbers below. For example, various open-source patches have been applied to this kernel, including Robert Love's preemption patch, Andrew Morton's low-latency patches and the O(1) scheduler from the 2.5 Linux tree. Other changes that might impact the performance of this test include algorithmic changes to reduce the remaining worst-case critical sections in the Linux kernel and changes to allow bottom-half interrupt processing to be performed inside of a kernel dæmon, whose scheduling policy and priority can be specified.

Figure 1 compares the interrupt response measured under RedHawk Linux using a shielded CPU and without using a shielded CPU. The difference between these runs is striking. In both test cases, most of the time the system was able to respond to the RTC interrupt in less than 100 microseconds. This shows that, in general, Linux responds to an interrupt in a timely manner. However, as stated above, the most important aspect of system metrics for a real-time system is the worst-case timings. This is because the worst cases are examples of system behavior that can cause a real-time application to miss its deadline.

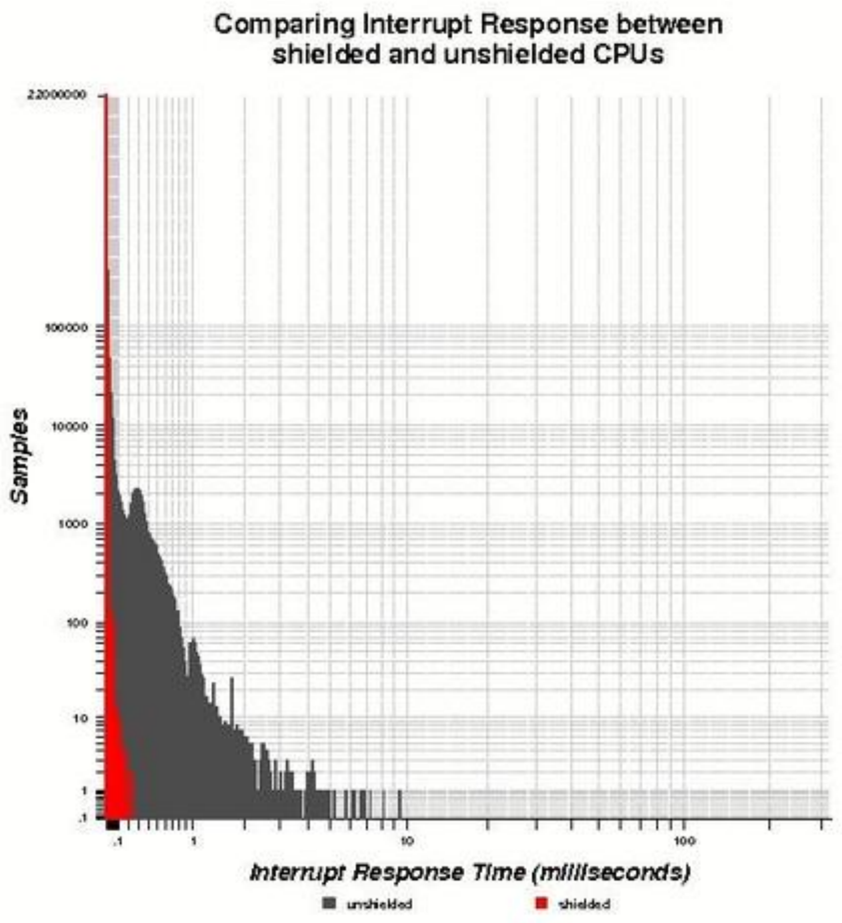


Figure 1. Comparing Interrupt Response between Shielded and Unshielded CPUs

In the shielded CPU case, the worst-case interrupt response time for the RTC interrupt was 220 microseconds. In the case where CPU shielding was not used, all interrupts responded in less than 10 milliseconds, an order of magnitude worse than the worst-case interrupt response time on a shielded CPU. Although less than one percent of the samples in this test case were greater than 200 microseconds, in many thousands of cases the interrupt response exceeded 500 microseconds. In a real-time system, each of these cases would be an opportunity for a missed deadline.

The same interrupt response test also was run on an unmodified 2.4.21 kernel.org kernel (Figure 2) as well as on Red Hat version 8.0 (Figure 3). This Red Hat kernel does not contain the preemption patch, but it does contain the low-latency patches, which are meant to address the longest critical sections in the

Linux kernel. Because shielded CPUs are not present in either of these kernels, the results are reported only for the non-shielded case.

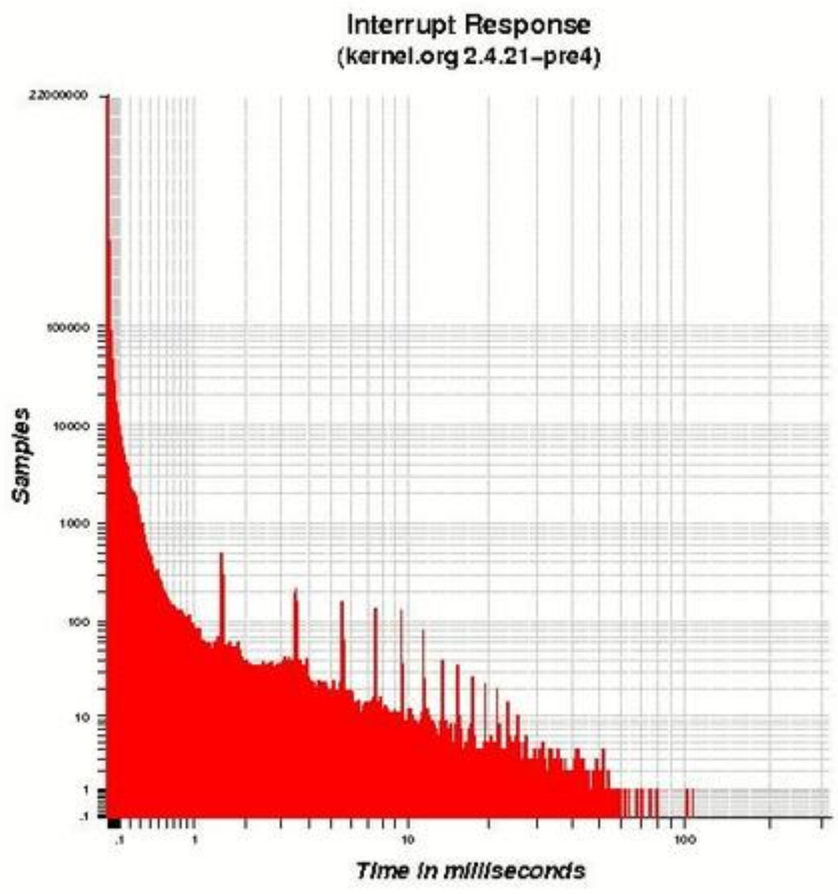


Figure 2. Interrupt Response (kernel.org 2.4.21-pre4)

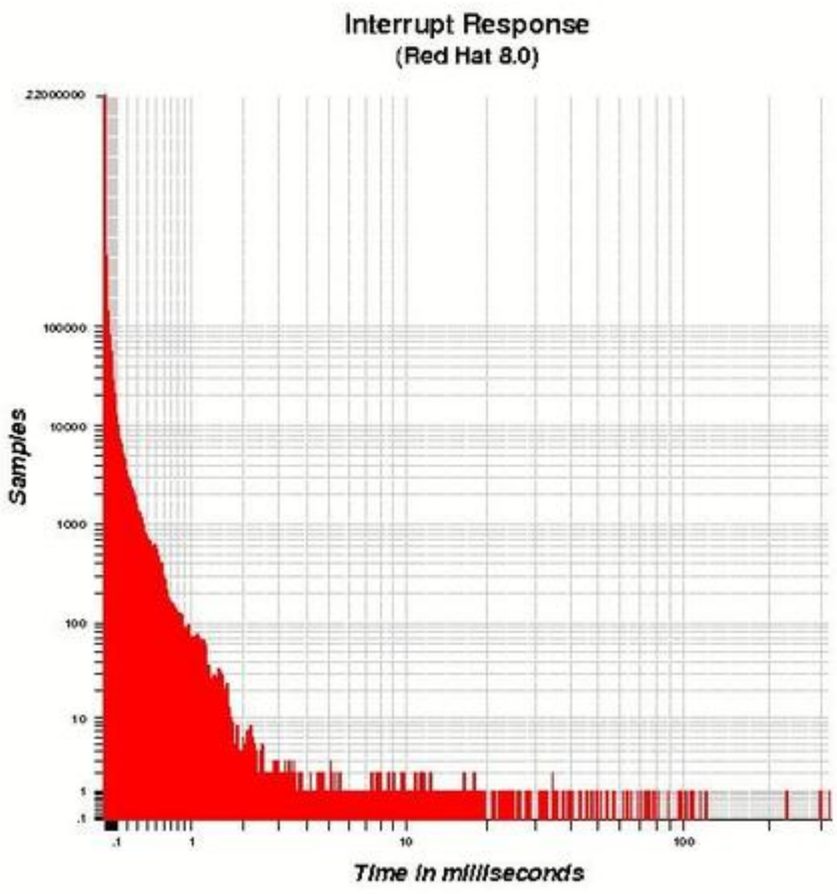


Figure 3. Interrupt Response (Red Hat 8.0)

These kernels show a typical interrupt response time similar to that measured on the RedHawk kernel, with most interrupts occurring in less than 100 microseconds. However, the worst-case interrupt response for these kernels is even worse than the non-shielded case under RedHawk Linux, with kernel.org showing a worst-case interrupt response of 107 milliseconds and Red Hat showing a worst-case interrupt response of 323 milliseconds. These results are not surprising considering that these kernels are tuned to achieve fairness between the processes that share the system and for general system throughput rather than for guaranteed real-time response.

Conclusions

It has been shown that a shielded CPU offers a significant improvement in the worst-case interrupt response time for a Linux system. Shielded CPUs are effective because they reserve critical computing resources for the highest priority tasks in the system. This is accomplished without affecting the standard application programming interface of Linux.

This article has discussed only the response to the RTC interrupt; it was chosen because it is a standard feature in most Linux implementations. It is possible, however, to achieve even better interrupt response guarantees by using other interrupt sources and more highly optimized device drivers. For a more extensive exploration of the shielded CPU concept as well as test results for a device driver that provides an even better interrupt response guarantee, see the whitepaper at www.ccur.com/isddocs/wp-shielded-cpu.pdf.

Stephen Brosky is Chief Scientist of the Integrated Solutions Division of Concurrent Computer Corporation. He was also a member of the IEEE committee that developed the POSIX 1003.1b and 1003.1c standards for real-time application interfaces and threads interfaces.

Advanced search

At the Forge

Blosxom

Reuven M. Lerner

Issue #121, May 2004

No modules, no SQL, no hassle. Create a blog with power features without even restarting the Web server.

Weblogs, or blogs, have grown dramatically in popularity over the past few years. Only a few people wrote blogs in the mid- and late 1990s, but now the blogging phenomenon is an overwhelming trend. Indeed, blogging is becoming so widespread that the *New York Times Magazine* published an article about it earlier this year—concentrating on high-school students who write their own blogs.

For example, as I write this, the Democratic primaries currently are in high gear, and every candidate has at least one official Weblog. Professional and armchair political commentators have set up their own blogs to analyze and counter claims the candidates make in their blogs and elsewhere.

Last month, we looked at COREBlog, a Zope product that makes it easy to create your own Weblog. Of course, COREBlog requires that you have a copy of Zope at your disposal and that you can install and modify products. Not everyone has this luxury, whereas almost every Web hosting provider makes it possible to run CGI programs written in Perl on your Web site. For this reason, many of the most popular blogging packages are small programs that do not raise the ire of an ISP.

This month, we look at Blosxom (pronounced blossom), a Weblog package written in Perl and designed to be run as a CGI program on a Web site. Blosxom was written by Rael Dornfest, a programmer at O'Reilly and Associates. I initially wrote off Blosxom as an unrealistic tool for blogging, assuming that its small size was indicative of its abilities. But Blosxom's power is not only in its strong feature set but in the way it allows us to mix and match functionalities.

Installation

Installing Blosxom should be a piece of cake for anyone with experience working with a Web server. It consists of a single CGI program written in Perl. In my case, all I had to do was copy the file, blosxom.cgi, to /usr/local/apache/cgi-bin, and I was up and running.

Of course, every piece of software requires at least a bit of configuration, and Blosxom is no exception. All of the configuration is handled by a few Perl variables at the top of the program. Comments make the purpose of each variable relatively clear. To configure Blosxom for my system, for example, I changed the following variables:

- $blog_title: the title of the Weblog as it appears to users and in the RSS syndication feed.
- $blog_description: blog description that appears on the front page and in the RSS feed.
- $datadir: each entry in a Blosxom Weblog actually is a text file on disk somewhere; $datadir defines where those files should reside.

With those three elements defined, my Weblog was up and running.

Adding Entries

I tested Blosxom by creating a simple text file in $datadir, introduction.txt:

```
This is a test entry.

<p>Hello!</p>
```

As Weblog entries go, this one was pretty boring. But it was interesting to see how this entry appears in my Weblog, preceded by a date, followed by a timestamp and a permanent link and with the first line boldfaced, as if it were a headline or title.

In other words, you can add entries to a Blosxom Weblog simply by creating new text files in the data directory. Any file ending with the value of $file_extension, which is txt by default, is considered a Weblog entry. This way, Emacs backup files, which end with ~, never are considered entries. But, if you are like me and have the habit of saving often while writing, you might be surprised to discover that your Weblog is being updated as you write it, live and for the whole world to see. If you want to work in the background, simply leave the .txt extension off the filename until you're ready to publish it.

On my workstation, where I installed Blosxom in the main cgi-bin directory, I can see my Blosxom blog as http://localhost/cgi-bin/blosxom.cgi.

Blosxom assigns a date and time to an entry based on the timestamp of the file that was created. Because I created the file on February 11, at 4PM, the Weblog entry was timestamped with that time. This means you can change the timestamp of a file retroactively with the touch command, as in:

```
   touch -t 200401011500 testing.txt
```

The above command modifies the date of the file testing.txt to 3PM on January 1, 2004. (If testing.txt does not exist already, it is created.) Although this might go against the etiquette of the Weblog universe, it certainly is possible.

More interestingly, you can modify the time of a Weblog entry to be in the future, using the same touch command on the command line. If the $show_future_entries configuration variable is set to 1, entries with such future dates are displayed all of the time. But in the default configuration, entries are displayed only when their date matches the current date. This means you can time-bomb your entries to be displayed on a particular time and date.

Flavours

If this were all that Blosxom provides, I would not be too impressed. But after examining it a bit more closely, I see that it contains a great deal of power. That power is there thanks to the combination of display templates (known as flavours, using the British spelling) and the ability to accept any number of plugin programs. The combination of these two features makes Blosxom quite extensible.

Blosxom comes with two flavours built-in, the default HTML flavour and the optional RSS flavour for the RSS syndication feed. You can view the RSS feed yourself by tacking `?flav=rss` onto the end of your blog's URL. So, if you normally view your Weblog at http://localhost/cgi-bin/blosxom.cgi, you can view the RSS feed for the site at http://localhost/cgi-bin/blosxom.cgi?flav=rss. Alternatively, you can specify your preferred flavour by changing the suffix of the page you retrieve. Thus, we can see RSS with http://localhost/cgi-bin/ blosxom.cgi/index.rss.

A complete flavour registry is available on the Blosxom Web site. But the basic idea is easy to grasp: in your data directory, alongside your Weblog entries, you create an HTML file whose name reflects the part of Blosxom's output you want to change.

The filename's suffix is the same as the flavour you want to modify. Thus, the file header.html changes the way the Weblog's header is displayed in the HTML flavour, and date.blah changes Blosxom's display of dates in the blah flavour. Users can set the flavour in the URL by adding the flav name-value pair (as we saw before), and the default is set in blosxom.cgi itself, with the variable $default_flavour. Because blog entries have a .txt suffix, you cannot have a txt flavour.

Each flavour file consists of an HTML snippet, along with Perl variable names that might be instantiated into the particular file. For example, story flavour files receive the variables $title and $body, among others. (A full list is available on the Blosxom Web site.) I thus can change my blog's output such that headlines are huge and right-aligned, followed by the body:

```
<p>
<H1 align="right">$title</h1>
<br />
$body
</p>
```

The above flavour inserts the $body variable, the contents of our blog story, verbatim into the HTML. This is fine if the blog author knows HTML and is willing to enter paragraph tags manually. But if we want to let people separate paragraphs with blank lines, we need to run a program on our story. Luckily, Blosxom makes it easy to write such programs with an extensible plugin architecture.

Plugins

Each plugin is a Perl program loaded with the require function, which reads and evaluates code in a particular file. So `require foo.pl` opens foo.pl and evaluates the code it contains. I normally suggest that people avoid `require` in favor of `use`, which executes a number of commands, including require. However, because require executes at runtime, whereas use executes during the compilation phase, it is far easier to work with it here.

Blosxom assumes that any file in the plugin directory, defined by the optional $plugin_dir variable, is a plugin. Plugins are both loaded and applied in alphabetical order, which means if you want to make sure a particular plugin is applied first or last, you might need to rename it.

Each plugin is nothing more than a simple Perl program that defines one or more subroutines. Every plugin must define the start subroutine, which simply returns 1. This allows Blosxom to determine that the plugin is alive, ready and willing to be invoked. A number of other plugin subroutines are available that each plugin optionally may define, ranging from entries (which returns a list of entries) to story (which allows you to modify the contents of a story). By breaking things down in this way, Blosxom allows for a tremendous amount of customization and sophistication, while keeping the core code small and compact.

So, what sorts of features can plugins provide? There seems to be only a few restrictions. You can change the source from which Weblog entries are retrieved, the way in which this list of entries is filtered, the templates used to display the entries and the contents of the entries themselves.

A large number of plugins are available from the Blosxom Web site. Some of them depend on other plugins, while others, such as the calendar, appear only if you are using a flavour that supports the plugin. Other plugins work immediately and merely need to be dropped into your plugin directory.

A simple example of a plugin that works out of the box is atomfeed, which provides an Atom syndication feed. Atom is a competitor to RSS that has been promoted by a number of heavy-hitting bloggers and programmers, in no small part because of the competing standards now evident in the RSS world. To get an Atom feed, simply copy the atomfeed plugin to your plugins directory. You then can retrieve your Atom feed with http://localhost/cgi-bin/blosxom.cgi?flav=atom or http://localhost/cgi-bin/blosxom.cgi/index.atom.

Writing Plugins

Listing 1 contains a simple filter, called egotrip, to make my name appear in boldface whenever it appears in a Weblog entry. Notice how the plugin must define its own package; this ensures that each plugin's subroutines are kept in a separate namespace and makes it possible for Blosxom to determine whether a package contains a particular method name.

Listing 1. egotrip.pl

```
#!/usr/bin/perl

use strict;
use warnings;
use diagnostics;

package egotrip;

# Returns 1 to indicate the plugin is active
sub start
{
    return 1;
}

# Boldfaces my name
sub story {
    my ($pkg, $path, $filename, $story_ref,
        $title_ref, $body_ref) = @_;

    $$body_ref =~ s|Reuven|<b>Reuven</b>|g;
    1;
}

1;
```

The actual work is done in the story subroutine, which is passed six arguments when invoked by Blosxom, corresponding to a number of items having to do with the entry. In our case, we care about changing only the body of the entry, which is in the final variable, known as $body_ref. As its name implies, this is a scalar reference, which means we can access or modify its contents by dereferencing it, using two $$ signs. With that in mind, it should not come as a surprise that we can boldface every instance of my name with:

```
$$body_ref =~ s|Reuven|<b>Reuven</b>|g;
```

Of course, we could make this step even more sophisticated and insert automatic hyperlinks to a number of different items:

```
$$body_ref =~ s|(Reuven Lerner)|
↪<a href="http://www.lerner.co.il/">$1</a>|g;
$$body_ref =~ s|(Linux Journal)|
↪<a href="http://www.linuxjournal.com/">$1</a>|g;
```

Indeed, a plugin of this sort already exists; it automatically creates links to the community-driven Wikipedia. Any text placed within [[brackets]] automatically is turned into a link to that on-line reference book.

Notice how flavours are HTML templates into which we can instantiate Perl variable values, whereas plugins are Perl programs. This division between display and actions takes a little bit of time to grasp, but it shouldn't be too difficult.

As for our paragraph-separating problem from before, there's no need to reinvent the wheel. You simply can download a plugin, Blox, that allows you to separate paragraphs with blank lines when writing your blog entry. The plugin then separates paragraphs with the HTML of your choice. Blox is listed on Blosxom's plugin registry (see the on-line Resources section).

The fact that Blosxom keeps all entries and flavours in a single directory is a bit disturbing to me and makes me wonder about the program's scalability. Even if my filesystem and Perl can handle that many files without too much trouble, do I really want to wade through them all? If and when this becomes a problem, an entries plugin probably can provide the right solution, scooping up files from multiple directories and returning an appropriate hash to Blosxom.

Conclusion

Blosxom is a powerful tool for creating a Weblog; it's more than it might appear at first glance. Blosxom consists of an easy-to-install, easy-to-configure CGI program written in Perl, but its true power lies in the fact that it lets you change every part of the display through a combination of flavours (display templates) and plugin routines. By mixing and matching existing flavours and templates with something of your own, it can be easy to create your own Weblog.

Resources for this article: /article/7454.

Reuven M. Lerner, a longtime consultant in Web/database programming, now is a graduate student in Learning Sciences at Northwestern University in Evanston, Illinois. You can reach him at reuven@lerner.co.il.

Archive Index Issue Table of Contents

Advanced search

Kernel Korner

Using DMA

James Bottomley

Issue #121, May 2004

DMA makes I/O go faster by letting devices read and write memory without bothering the CPU. Here's how the kernel keeps track of changes that happen behind the CPU's back.

DMA stands for direct memory access and refers to the ability of devices or other entities in a computing system to modify main memory contents without going through the CPU. The desirability of DMA lies in not troubling the CPU; the system simply can request that the data be fetched into a particular memory region and continue with other tasks until the data is ready. Most of the problems in DMA, however, are due to the lack of CPU involvement.

The problems with DMA are threefold. First, the CPU probably is operating a memory management unit. Therefore, the address the CPU uses to describe the memory region is not the same as the physical address of main memory. Second, because the transfer is to main memory, the caches between that memory and the CPU probably are not coherent (see “Understanding Caching”, *LJ*, January 2004. Third, there also may be a memory management unit on the I/O bus (called an IOMMU). This means the bus address the device uses to transfer the data may not be the same as the physical memory address or the CPU's virtual memory address. This concept is alien to most x86 people. Even here, though, the use of GARTs (graphical aperture remapping tables) for the AGP bus is making the x86 refusal of IOMMUs less strong than it once was.

The API that manages DMA in the Linux kernel must take into account and solve all three of these problems. In addition, because most DMA is done from devices on an external bus, three additional problems may occur. First, the I/O device addressing width may be different from the address width of physical memory. For instance, an ISA device is limited to addressing 24 bits, and some PCI devices in 64-bit systems are limited to addressing 32 bits. Second, the I/O bus controller circuitry itself may cache requests. This occurs mainly on the PCI bus, where write requests may be held in the PCI controller in the hope that it may accumulate them for rapid transfer to the device. This phenomenon is called PCI posting. Third, the operating system may request a transfer to a region that is contiguous in its virtual memory space but fragmented in the memory's physical space, usually because the requested transfer crosses multiple pages. Such a transfer must be accomplished using scatter/gather (SG) lists.

This article deals strictly with the DMA API for devices. The new generic device model in Linux 2.6 provides a nice way of describing device characteristics and finding their bus properties using a hierarchical tree. The interfaces described have undergone considerable revision in the transition from 2.4 to 2.6. Although the general principles of this article apply to 2.4, the API described and the kernel capabilities apply only to the 2.6 kernel.

SG Lists

For any DMA transfer, the first problem to consider is the user may request a large transfer (kilobytes to megabytes) to a given buffer. Because of the way virtual memory is managed, however, this area, which is contiguous in virtual space, may be composed of a sequence of pages fragmented all over physical memory. Linux expects that any transfer above a page size (4KB on an x86 system) needs to be described by an SG list. Ordinarily, these lists are constructed by the block I/O (BIO) layer. A key job of the device driver is to parameterize the BIO layer in the way it may divide up the I/O into SG list elements.

Almost every device that transfers large amounts of data is designed to accept these transfers as some form of SG list. Although the exact form of this list is likely to differ from the one supplied by the kernel, conversion usually is trivial.

I/O Memory Management Units (IOMMUs)

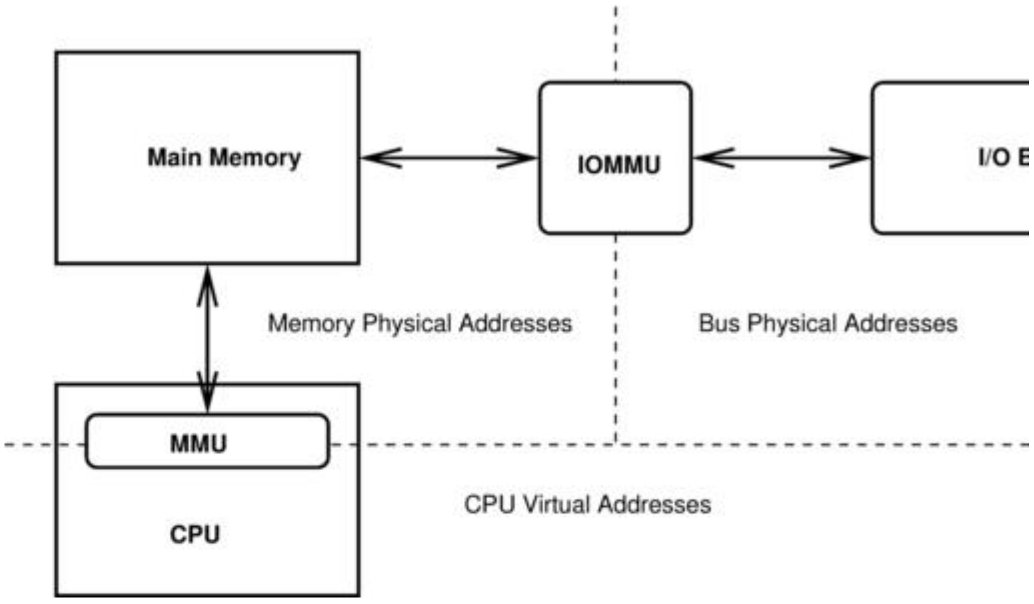


Figure 1. Address Domains in DMA

An IOMMU is a memory management unit that goes between the I/O bus (or hierarchy of buses) and the main memory. This MMU is separate from the IOMMU built in to the CPU. In order to effect a transfer from the device to main memory, the IOMMU must be programmed with the address translations for the transfer in almost exactly the same way as the CPU's MMU would be programmed. One of the advantages of doing this is an SG list generated by the BIO layer can be programmed into the IOMMU such that the memory region appears to be contiguous again to the device on the bus.

GARTs and IOMMU Bypass

A GART basically is like a simple IOMMU. It consists of a window in physical memory and a list of pages. Its job is to remap physical addresses in the window to physical pages in the list. The window typically is narrow, only about 128MB or so, and any accesses to physical memory outside this window are not remapped. This insufficiency exposes a weakness in the way the Linux kernel currently handles DMA: none of the DMA APIs have a failure return for failing to map the memory. A GART has a limited amount of remapping space, however, and once that is exhausted nothing may be mapped until some I/O completes and frees up mapping space.

Sometimes, like a GART, an IOMMU may be programmed not to do address remapping between the I/O bus and the memory in certain windows. This is called bypass mode and may not be possible for all types of IOMMU. Bypass mode is desirable sometimes, because the act of remapping adds a performance hit to the transfer, so lifting the IOMMU out of the way can achieve an increase in throughput.

The BIO layer, however, assumes that if an IOMMU is present, it is being used, and it calculates the space needed for the device SG list accordingly. Currently, no way exists to inform the BIO layer that the device wishes to bypass the IOMMU. A problem occurs if the BIO layer assumes the presence of an IOMMU; it also assumes SG entries are being coalesced by the IOMMU. Thus, if the device driver decides to bypass the IOMMU, it may find itself with more SG entries than the device allows.

Both of these issues are being worked on in the 2.6 kernel. A fix for the IOMMU bypass already is under consideration and will be invisible to driver writers, because the platform code will choose when to do the bypass. The fix for the inability to map probably will consist of making the mapping APIs return failure. Because this fix affects every DMA driver in the system, implementing it is going to be slow.

Bus Widths and DMA Masks

In order to communicate the maximum addressing width, every generic device has a parameter, called the DMA mask, that contains a map of set bits corresponding to the accessible address lines that must be set up by the device driver. The DMA width has two separate meanings depending on whether an IOMMU is in use. If there is an IOMMU, the DMA mask simply represents a limitation on the bus addresses that may be mapped, but through the IOMMU, the device is able to reach every part of physical memory. If there is no IOMMU, the DMA mask represents a fundamental limit of the device. It is impossible for the device to transfer to any region of physical memory outside this mask.

The block layer uses the DMA mask when building a scatter/gather list to determine whether the page needs to be *bounced*. By bounce, I mean the block layer takes a page from a region within the DMA mask and copies all the data to it from the out-of-range page. When the DMA has completed, the block layer copies it back again to the out-of-range page and releases the bounce page. Obviously, this copying back and forth is inefficient, so most manufacturers try to ensure that the devices with which their server-type machines ship don't have DMA mask limitations.

DMA and Coherency

DMA occurs without using the CPU, so the kernel has to provide an API to bring the CPU caches into sync with the memory changed by the DMA. One thing to remember is the DMA API brings the CPU caches up to date only with respect to the kernel virtual addresses. You must use a separate API, described in my article “Understanding Caching”, to update the caches with respect to user space.

Bus Posting (Caching)

Sometimes high-end bus chips also come with caching circuitry. The idea behind this is that writes from the CPU to the chipset are fast, but writes across the bus are slow, so if the bus controller caches the writes, the CPU doesn't need to wait for them to complete. The problem with bus posting, as this type of caching is called, is that no CPU instruction is present to flush the bus cache, so bus cache flushes work according to a strict set of rules to ensure proper ordering. First, the rules are that only memory-based writes may be cached. Writes that go through I/O space are not cached. Second, the ordering of memory-based reads and writes must be preserved strictly, even if the writes are cached. This last property allows a driver writer to flush the cache. If you issue a memory-based read to any part of the device's memory region, all cached writes are guaranteed to be issued before the read begins.

No API is available to help with posting, so driver writers need to remember to obey the bus posting rules when reading and writing a device's memory region. A good trick to remember is if you really can't think of a necessary read to flush the pending writes, simply read a piece of information from the device's bus configuration space.

Using the DMA API in a Device Driver

The API is documented thoroughly in the kernel documentation directory (Documentation/DMA-API.txt). The generic DMA API also has a counterpart that applies only to PCI devices and is described in Documentation/DMA-mapping.txt. The intent of this section is to provide a high-level overview of all the steps necessary to get DMA working correctly. For detailed instructions, you also should read the above-mentioned documentation.

To start, when the device driver is initialized, the DMA mask must be set:

```
int
dma_set_mask(struct device *dev, u64 mask);
```

where `dev` is the generic device and `mask` is the mask you are trying to set. The function returns true if the mask has been accepted and false if not. The mask may be rejected if the actual system width is narrower; that is, a 32-bit system may reject a 64-bit mask. Thus, if your device is capable of addressing all 64 bits, you first should try a 64-bit mask and fall back to a 32-bit mask if setting the 64-bit mask fails.

Next, you need to allocate and initialize the queue. This process is somewhat beyond the scope of this article, but it is documented in Documentation/block/. Once you have a queue, two vital parameters need to be adjusted. First, allow for the largest size of your SG table (or tell it to accept an arbitrarily big one) with:

```
void
blk_queue_max_hw_segments(request_queue_t *q,
                       unsigned short max_segments);
```

Second, (if you need it), the overall maximum size:

```
void
blk_queue_max_sectors(request_queue_t *q,
                      unsigned short max_sectors);
```

Finally, the DMA mask must be programmed into the queue:

```
void
blk_queue_bounce_limit(request_queue_t *q,
                       u64 max_address);
```

Usually, you set max_address to the DMA mask. If an IOMMU is being used, however, max_address should be set to BLK_BOUNCE_ANY to tell the block layer not to do any bouncing.

Device Operation

To operate a device, it must have a request function (see the BIO documentation) whose job it is to loop around and pull requests from the device queue using the command:

```
struct request
*elv_next_request(request_queue_t *q);
```

The number of mapping entries required by the request are located in req->nr_phys_segments. You need to allocate an interim table of this size in units of sizeof(struct scatterlist). Next, do the interim mapping with:

```
int
blk_rq_map_sg(request_queue_t  *q,
              struct  request  *req,
              struct scatterlist *sglist);
```

This returns the number of SG list entries used.

The following command provides the interim table supplied by the block layer, which finally is mapped using:

```
int
dma_map_sg(struct  device  *dev,
           struct scatterlist *sglist, int count,
           enum dma_data_direction dir);
```

where `count` is the value returned and `sglist` is the same list passed into the function blk_rq_map_sg. The return value is the number of actual SG list entries needed by the request. The SG list is reused and filled up with the actual entries that need to be programmed into the device's SG entries. The dir provides a hint about how to cope correctly with cache coherency. It can have three values:

1. DMA_TO_DEVICE: the data is being transferred from memory to the device.
2. DMA_FROM_DEVICE: the device is transferring data into main memory only.
3. DMA_BIDIRECTIONAL: no hint is given about the transfer direction.

Two macros should be used when traversing the SG list to program the device's SG table:

```
dma_addr_t
sg_dma_address(struct scatterlist *sglist_entry);
```

and:

```
int
sg_dma_len(struct scatterlist *sglist_entry);
```

which return the bus physical address and segment lengths, respectively, of each entry.

The reason for this two-stage mapping of the request is because the BIO layer is designed to be generic code and has no direct interaction with the platform layer, which knows how to program the IOMMU. Thus, the only thing the BIO layer can calculate is the number of SG segments the IOMMU makes for the request. The BIO layer doesn't know the bus addresses the IOMMU assigns to these segments, so it has to pass in a list of the physical memory addresses of all the pages that need to be mapped. It is the dma_map_sg function that communicates with the platform layer, programs the IOMMU and retrieves the bus physical list of addresses. This too is why the number of elements the BIO layer needs for its list may be longer than the number returned by the DMA API.

When the DMA has completed, the DMA transaction must be torn down with:

```
int
dma_unmap_sg(struct  device *dev,
             struct scatterlist *sglist,
             int hwcount,
             enum dma_data_direction dir);
```

where all the parameters are the same as those passed into dma_map_sg except for hwcount, which should be the value returned by that function. And finally, the SG list you allocated may be freed and the request completed.

Accessing Data in the DMA region

Usually, the device driver operates without touching any of the data it is transferring. Occasionally, however, the device driver may need to modify or inspect the data before handing it back to the block layer. To do this, the CPU caches must be made coherent with the data by using:

```
int
dma_sync_sg(struct device *dev,
            struct scatterlist *sglist,
            int hwcount,
            enum dma_data_direction dir);
```

where the arguments are identical to dma_unmap_sg.

The most important factor in accessing data is when you do it. The rules for accessing depend on dir:

- DMA_TO_DEVICE: the API must be called after modifying the data but before sending it to the device.
- DMA_FROM_DEVICE: the API must be called after the device has returned the data but before the driver attempts to read it.
- DMA_BIDIRECTIONAL: the API may need to be called twice, after modifying the data but before sending it to the device and after the device finishes with it but before the driver accesses it again.

Allocating Coherent Memory

Most devices use mailbox-type regions of memory for communication between the device and the driver. The usual characteristic of this mailbox region is that it is never used beyond the device driver. Managing the coherency of the mailbox using the previous API would be quite a chore, so the kernel provides a method for allocating a region of memory guaranteed to be coherent at all times between the device and the CPU:

```
void
*dma_alloc_coherent(struct device *dev, size_tsize,
                    dma_addr_t *physaddr, int flag);
```

This returns the virtual address of a coherent region of size that also has a bus physical address (physaddr) to the device. The flag is used to specify the allocation type GFP_KERNEL to indicate the allocation may sleep to obtain the memory and GFP_ATOMIC to indicate the allocation may not sleep and may return NULL if it cannot obtain the memory. All memory allocated by this API also is guaranteed to be contiguous both in virtual and bus physical memory. There is an absolute requirement that size be less than 128KB.

As part of driver removal, the coherent region of memory must be freed with:

```
void
dma_free_coherent(struct device *dev, size_tsize,
                  void *virtaddr,
                  dma_addr_t *physaddr);
```

where `size` is the size of the coherent region and `virtaddr` and `physaddr` are the CPU virtual and bus physical addresses, respectively, returned for the coherent region.

Conclusions

The article offers a lightning-quick overview of how the block layer interacts with device drivers to produce SG lists for programming devices. You may find several additional pieces of the DMA API useful, including APIs that handle unfragmented regions of physical memory. If this article whets your appetite, you're now ready to move on to reading the kernel Documents and source.

James Bottomley is the Software Architect for SteelEye. He also is an active member of the Open Source community. He maintains the SCSI subsystem, the Linux Voyager port and the 53c700 driver and has made contributions to PA-RISC Linux development in the area of DMA/device model abstraction.

Cooking with Linux

Eye Candy for Admins? Aye, SuperKaramba!

Marcel Gagné

Issue #121, May 2004

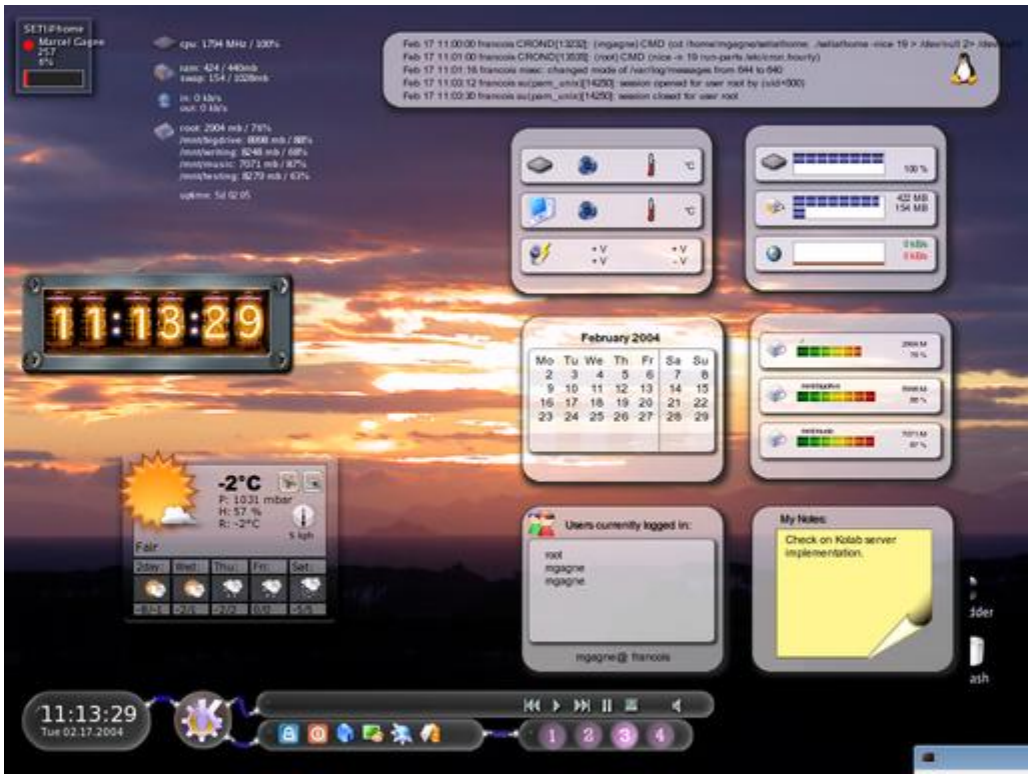
Don't merely monitor your system logs—give your system stats displays a certain *je ne sais quoi* with GUI tools.

You see, François, administering your Linux system is all about information. When it comes down to knowing what is happening with our servers, too much information is just about right. Yes, *mon ami*, I am joking, but only a little. Every Linux system has an ongoing chatter, whether it is a server or a desktop. Statistics constantly are pouring in on CPU activity, disk space and memory allocation and logs are being filled. Don't forget logs, François, logs of e-mail traffic, FTP and Web site transfers, services starting and stopping. That's a lot to keep up with, and having the right tools is essential.

Quoi? It looks like eye candy? Well, it is, François. No one said keeping track of what your systems are up to couldn't be fun, not to mention stylish. But enough of this. Our guests will be here any moment, and we must be ready. *Mon Dieu!*, they are here already. Welcome, everyone, to *Chez Marcel*, home of fine Linux fare and excellent wines. Please sit and I will have François fetch the wine. To the wine cellar, François! Bring back the 2000 Chablis Les Clos. *Vite!*

Make yourselves comfortable *mes amis*. The theme of this issue is system administration, and any one of us who runs any kind of computer—even if it's only a home system—is the administrator of that system. You are the boss, *mes amis*. Sometimes, you are the boss of many, and sometimes you are your own boss. Now you all know what they say about system administrators, *non*? A good system administrator is forewarned, and as the saying goes, forewarned is forearmed. Four arms, of course, is a highly unusual number of arms for a sysadmin to have; although many, I'm sure, could see the advantage. It is because of this inherent strangeness that I present you with today's menu, a collection of graphical tools that should keep you informed while adding a certain *je ne sais quoi* to your system.

Desktop wallpaper is interesting enough, but dynamic applications can be put on the desktop as well. For instance, imagine a monitor for CPU usage, disk space and network activity floating transparently on your desktop, constantly being updated. If this sounds interesting, get your hands on Adam Geitgey's SuperKaramba. The Super in front of Karamba might lead you to believe there was a predecessor to this package, and you would be right. Hans Karlsson is the author of the original Karamba. To see a wonderfully busy SuperKaramba desktop, have a look at Figure 1.



```
Feb 17 11:00:00 francois CROND[13232]: (mgagne) CMD (cd /home/mgagne/setiathome; ./setiathome -nice 19 > /dev/null 2> /dev
Feb 17 11:01:00 francois CROND[13508]: (root) CMD (nice -n 19 run-parts /etc/cron.hourly)
Feb 17 11:01:16 francois msec: changed mode of /var/log/messages from 644 to 640
Feb 17 11:03:12 francois su(pam_unix)[14250]: session opened for user root by (uid=500)
Feb 17 11:03:30 francois su(pam_unix)[14250]: session closed for user root
```

Figure 1. SuperKaramba themes are informative and look cool.

Pick up the latest source at the SuperKaramba Web site (see the on-line Resources section). Building the package is something with which most of you are familiar—the extract and build five-step:

```
tar -xzvf superkaramba-0.33.tar.gz
cd superkaramba-0.33
./configure --prefix=/usr
make
su -c "make install"
```

You need Python development libraries to build the package from source. For those who would prefer to skip all that compiling, links are provided on the main SuperKaramba page, along with precompiled binaries for several distributions. If you do decide to compile SuperKaramba and are running KDE 3.2, you may encounter a little weirdness. This also might be fixed in the source code by the time you read this, but themes are always on top, blocking other active windows. Fix this by editing the karamba.cpp file in the src directory after you extract the source. Look for the line that reads:

```
   KWin::setType(winId(), NET::Dock);
```

and comment it out by adding two slashes in front of the line, like this:

```
   // KWin::setType(winId(), NET::Dock);
```

That's it. Go ahead with your compile and all is well.

Start SuperKaramba from the command line with `superkaramba`. The program also shows up in my KDE Utilities menu. When the program starts, you see a window offering you three choices (Figure 2). *Mon Dieu!* François, bring the wine here *immï¿½iatement* and refill my glass. It seems that some Python did sneak into the code after all.

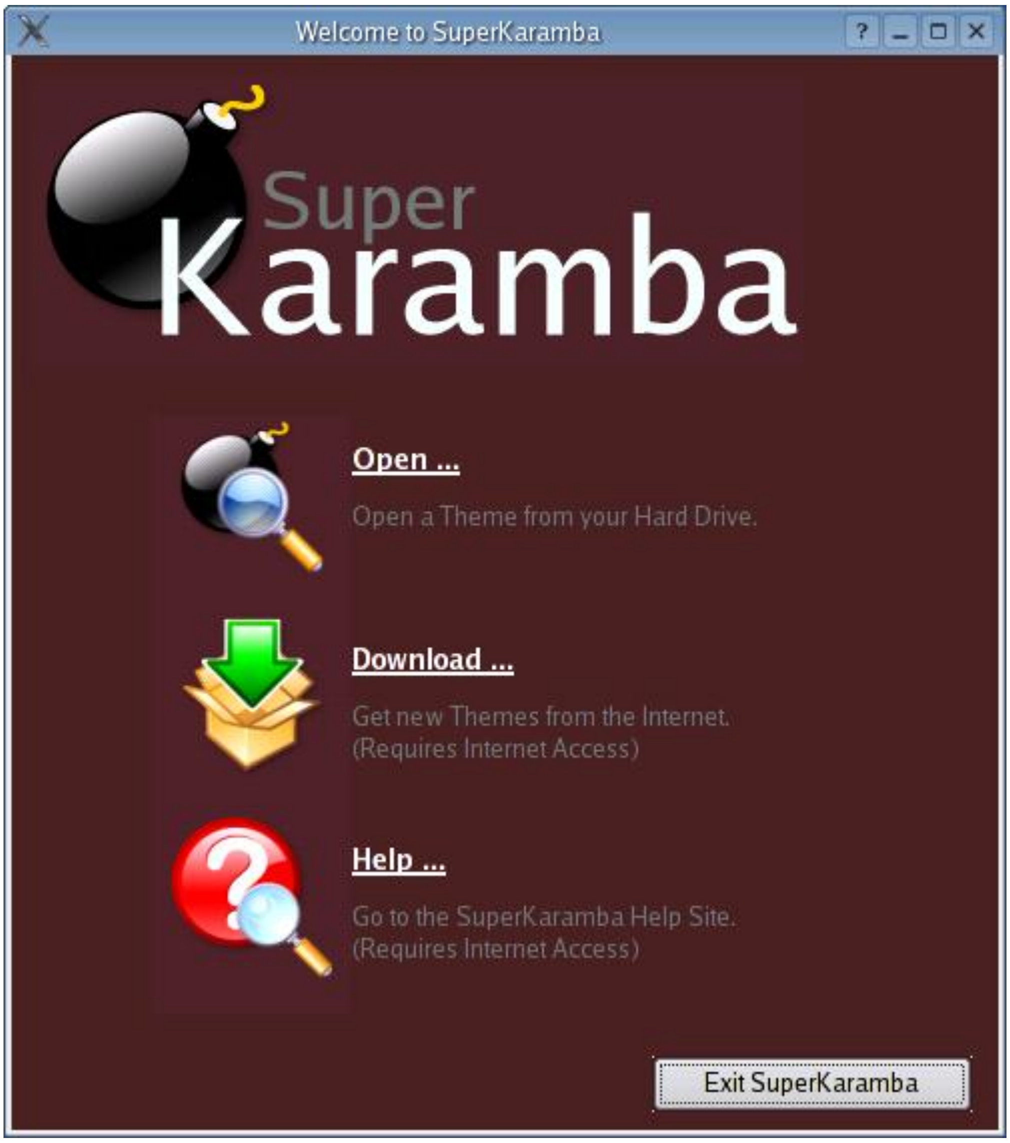
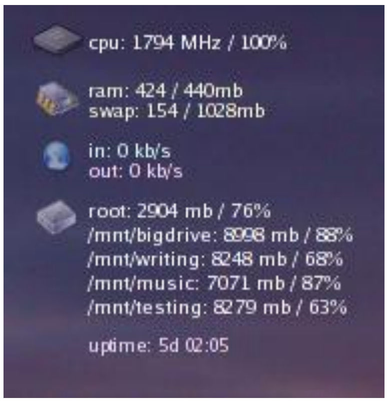

Welcome to SuperKaramba
Super Karamba
Open ...
Open a Theme from your Hard Drive.
Download ...
Get new Themes from the Internet. (Requires Internet Access)
Help ...
Go to the SuperKaramba Help Site. (Requires Internet Access)
Exit SuperKaramba

Figure 2. The SuperKaramba Startup Screen

Clicking Open lets you select from SuperKaramba themes already installed on your system. As of this writing, the themes section of the SuperKaramba site had not yet been launched. You instead were directed to the Karamba section of KDE-Look.org. Look for Karamba in the menu on the left. The various themes are arranged by several criteria you can select by clicking the tabs above the list. Browse by the most recent or most popular in terms of downloads or highest rated.

Each Karamba theme shows a screenshot and provides a download. Choose something appropriate, then download and extract the tarball to an appropriate directory. Some of these are tarred and gzipped, others are tarred and bzipped. There is no hard-and-fast rule as to where themes end up living, because the open dialog lets you find them anywhere. I created a directory called Karamba where I store my files. Let's say that Kelley at table 16 wanted to load Flavio Gargiulo's Micromon theme (Figure 3), a trimmed-down version of Simon Ask Ulsnes' Minimon, which comes in a tarred and gzipped bundle. He would do the following:

```
cd ~/Karamba_dir
tar -xvf 8722-micromon-1.0.tar.gz
```



```
cpu: 1794 MHz / 100%

ram: 424 / 440mb
swap: 154 / 1028mb

in: 0 kb/s
out: 0 kb/s

root: 2904 mb / 76%
/mnt/bigdrive: 8998 mb / 88%
/mnt/writing: 8248 mb / 68%
/mnt/music: 7071 mb / 87%
/mnt/testing: 8279 mb / 63%

uptime: 5d 02:05
```

Figure 3. Micromon systems stats float across your desktop.

On the other hand, if Jon at table 9 wanted to install Matti's Liquid Weather theme (Figure 4), which comes in a bzip2 tarred bundle, he would use this command:

```
cd ~/Karamba_dir
tar -xjvf lwp-1.9.tar.bz2
```

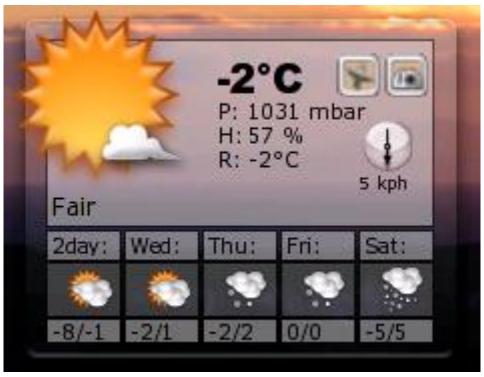
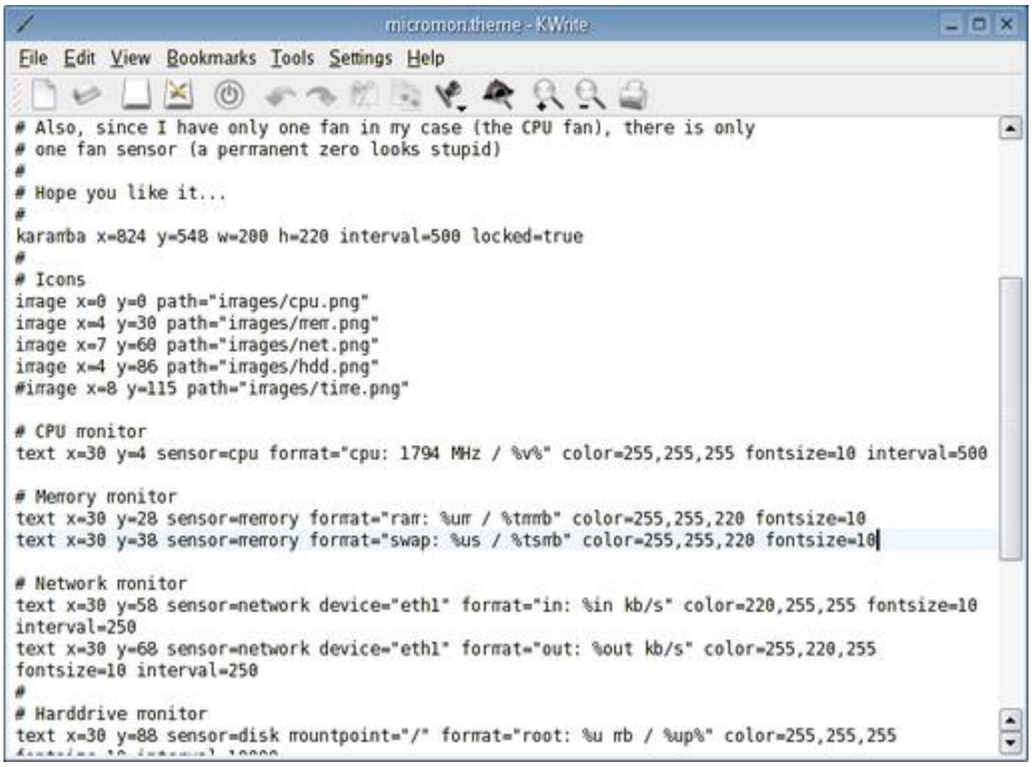


Figure 4. The Liquid Weather theme can be modified to report on any location.

Perhaps a weather theme isn't strictly sysadmin material, but that's Jon for you. In any case, there's no other building or installing. Extract the files, and you are done. Then, navigate to the install directory and look for the file with a .theme extension. Click that file and click OK. The theme starts up and appears on your desktop.

Launching subsequent themes doesn't require you to re-run `superkaramba`. Instead, right-click on a running theme and choose Open new theme. Right-clicking on a running theme provides several menu options, including the ability to edit a currently running theme or its configuration file (Figure 5). Theme files generally are easy to read and lend themselves to simple edits. For instance, I modified the Micromon theme (Figure 1) to display my own disk partitions instead of the ones defined by the author. I also increased the font size to make it easier to read.



```
# Also, since I have only one fan in my case (the CPU fan), there is only
# one fan sensor (a permanent zero looks stupid)
#
# Hope you like it...
#
karamba x=824 y=548 w=200 h=220 interval=500 locked=true
#
# Icons
image x=0 y=0 path="images/cpu.png"
image x=4 y=30 path="images/mem.png"
image x=7 y=60 path="images/net.png"
image x=4 y=86 path="images/hdd.png"
#image x=8 y=115 path="images/time.png"

# CPU monitor
text x=30 y=4 sensor=cpu format="cpu: 1794 MHz / %v%" color=255,255,255 fontsize=10 interval=500

# Memory monitor
text x=30 y=28 sensor=memory format="ram: %um / %tmmb" color=255,255,220 fontsize=10
text x=30 y=38 sensor=memory format="swap: %us / %tsmb" color=255,255,220 fontsize=10

# Network monitor
text x=30 y=58 sensor=network device="eth1" format="in: %in kb/s" color=220,255,255 fontsize=10
interval=250
text x=30 y=68 sensor=network device="eth1" format="out: %out kb/s" color=255,220,255
fontsize=10 interval=250
#
# Harddrive monitor
text x=30 y=88 sensor=disk mountpoint="/" format="root: %u mb / %up%" color=255,255,255
```

Figure 5. Editing a SuperKaramba .theme File

To move a theme around on your desktop, hold down the Alt key, left-click with the mouse and drag the SuperKaramba theme to wherever you want it located on your desktop. That positional information is stored in .rc files located in the .superkaramba directory in your home directory. Mine, for instance, is in /home/marcel/.superkaramba. Before you go looking at the .rc file for one of your running themes, I should warn you about an interesting effect. Until you shut down a theme or log out, the .rc file remains blank. One way to write out the file without shutting down the application is to right-click on the running theme and select Reload theme. Here's a sample from Ryan Nickell's skSeti desktop application, a small theme that monitors my SETI@home progress:

```
[config menu]
bgImage=false

[internal]
desktop=0
fastTransforms=true
lockedPosition=false

[theme]
Version=0.01
background=bg.png
firstTime=No
seti_Directory=/home/marcel/setiathome/
widgetHeight=100
widgetPosX=0
widgetPosY=0
widgetWidth=100
```

You should find a great number of SuperKaramba themes available. Some are Kicker panel replacements, like Sven Johannsen's Glass Machine (pictured on the bottom of Figure 1). Aside from giving you slick access to your kicker functions, the Glass Machine also makes all your XMMS controls handy. A little music helps sysadmins get their work done, and several jukebox and multimedia toys are available to choose from on the site.

Other themes are just plain fun, such as Reverant2501's TubeClock with seconds. Older readers should find a comforting nostalgia there, and the younger crowd simply may think it looks cool. More useful are packages such as Chip 2003's TDE (T Desktop Enhancements), which provides a number of tools, including a notepad and a log viewer, as well as disk, memory and performance monitors. Chip 2003 also provides another nice multimonitor, TMon. There's more, but let me leave you with one final monitor to experience. It's called The (as in the only) Karamba theme. It's from artoo, and it's a GkRellm-like monitor for SuperKaramba—everything you need to know in one vertical monitor.

All of these SuperKaramba improvements provide a means for you to stay informed and look good doing it. Once you start playing with these themes, you may become the most informed administrator out there. Unfortunately, you may not be the most productive—just a little joke, *mes amis*.

It appears that closing time quickly approaches—*Mon Dieu!* But I can see that you all are engrossed at your various desktops installing themes to suit your individual tastes. Perhaps François will be kind enough to refill your glasses once more while you experiment. Until next time, *mes amis*, let us all drink to one another's health. *A votre santé Bon appétit!*

Resources for this article: /article/7455.

Marcel Gagné (mggagne@salmar.com) lives in Mississauga, Ontario. He is the author of *Moving to Linux: Kiss the Blue Screen of Death Goodbye!* (ISBN 0-321-15998-5) from Addison Wesley. His first book is the highly acclaimed *Linux System Administration: A User's Guide* (ISBN 0-201-71934-7). In real life, he is president of Salmar Consulting, Inc., a systems integration and network consulting firm.

Archive Index Issue Table of Contents

Advanced search

EOF

Open Legal Research

Pamela Jones

Issue #121, May 2004

One little Web site about a batty lawsuit became the tech news hit of the year. And Internet research can help clear the next legal minefield too.

Open legal research isn't a phrase in law dictionaries. Law firms normally are secretive and keep everything very hush-hush. What we've been doing at Groklaw is pioneering work, what Open Source Risk Management's Daniel Egger called “a new kind of collaborative, real-time, you-can't-get-away-from-us legal resource”. The Internet makes it possible.

Like a lot of ideas that turn out really to work, it was supposed to be something else but morphed. When I started, it was only me, one geeky paralegal who didn't like what The SCO Group was trying to do to Linux and decided to help. SCO seemed to be pursuing a strategy of delaying any test of its claims in court, while at the same time maximizing fear, uncertainty and doubt about Linux. A lot of my early work was finding evidence to rebut SCO's public statements—anti-FUD, if you will. I also did detailed research that I hoped would be helpful and interesting.

Readers started to show up, first a trickle, then a flood. Groklaw today has more than 4,500 members and a readership in the millions. Some of the volunteers knew things I didn't, especially about the code issues, but they didn't realize what they knew was useful legally. I decided to show them how to do legal research and how to identify useful evidence, so I posted legal research guides and links to sites and explained terminology, such as “promissory estoppel” or “slander of title”, by quoting from lawyers and judges and providing links to more information.

Groklaw's readers include direct witnesses to or even central participants in events described in court filings and public statements. When SCOForum happened and the debate began on what the code was, I simply wrote to Dennis Ritchie and asked him. When SCO claimed to own C++, I wrote to its author, Dr Bjarne Stroustrup, and he denied it publicly. When they claimed the ABI files, Warrren Toomey from the UNIX Heritage Society wrote a rebuttal article. The experts are alive and able to testify.

My volunteer Webmaster, Peter Roozemaal, switched Groklaw to Geeklog software and wrote some needed features to facilitate collaborative work. We have groups for different projects. I invite some to take on more responsibilities. “Dr Stupid”, for example, is my primary lieutenant now on all code articles.

One group tracks court postings of legal documents. Others go to the courthouse for paper-only exhibits, while another group OCRs, another proofreads and others research. Another group works on the quotation database, under Leif Jensen. Still another group transcribes press conferences and other events. IBM cited the group's Groklaw work in support of a motion it later won.

Attorneys, including Eben Moglen, Dan Ravicher, Mark Webbink, Dennis Karjala, Webster Knight, Lewis Mettler, Anupam Chander and others have helped with articles, interviews and behind-the-scenes news.

To offer one example of group research and interacting with living experts, one group of more than a dozen, led by Alex Roston, collected evidence showing that an important contributor to the Linux kernel was Tigran Aivazian, at the time an “old SCO” employee. He worked on some of the high-end functionality issues involved in the IBM case. Aivazian not only reviewed the draft of the Groklaw article, he went on the record that his contributions were with the knowledge and permission of his supervisor, explaining in writing that he “requested permission from Wendy (Development Director) before the release under GPL and she confirmed that SCO has no claims to this work whatsoever and has no objections to its release under GPL”.

Another group led by Frank Sorenson did an article on Linux ABI files, which in my opinion undercut SCO's ability to sue end users successfully. It was a research volunteer, Rand McNatt, who discovered that Novell was registering UNIX copyrights, a story Groklaw broke.

What's Next?

The whole SCO drama is really a “shot across the bow” for GNU/Linux, I believe, and I am sure more nothing-to-lose companies will launch suits against free/open-source code. I'm already looking beyond SCO in my research for OSRM and Groklaw.

We have begun a new project, the Unix History Timeline Project. OSRM is donating a portion of my time to it. This is a comprehensive oral history and documentation project covering the important events in the licensing and ownership history of UNIX. More than 30 other flavors of proprietary UNIX, other than AT&T's, are available. This is a call for anyone who wishes to send me whatever you think might be relevant.

We are determined to find and clear any potential minefields, if they exist. Within 48 hours of first announcing the project, we had more than 400 volunteers, including most of the published historians of UNIX and many of the people who actually contributed to UNIX in the first place. One wrote that I am now “the maintainer of the Linux anti-lawsuit kernel”. That's a good description of what our project is all about. It's a nice compliment as well.

I also have asked Dan Ravicher of the Public Patent Foundation to cover patents for Groklaw in the future. Our method obviously is well suited to finding prior art. If other attorneys wish to contribute to Groklaw, please contact me.

People are hungry to understand legal news, and they want to help. I believe open legal research works and that there will be many more projects like ours in the future. Groklaw is the proof of concept.

Pamela Jones is the Founder and Editor of Groklaw, which was launched in May 2003. She has just been named Director of Litigation Risk Research for Open Source Risk Management (OSRM), which is currently preparing to offer comprehensive vendor-neutral Open Source Defense Insurance to GNU/Linux users.

Archive Index Issue Table of Contents

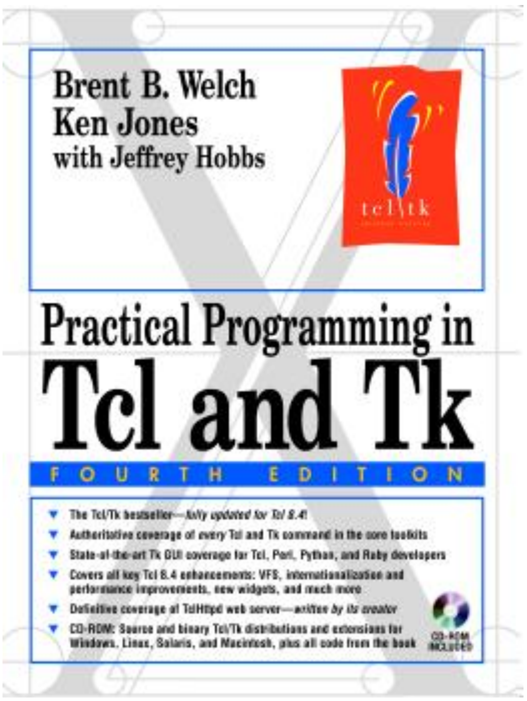
Advanced search

Practical Programming in Tcl and Tk, Fourth Edition, by Brent B. Welch, Ken Jones and Jeffrey Hobbs

Marty Leisner

Issue #121, May 2004

One of the outstanding points is that this book spells out in which versions newer features were introduced.


Brent B. Welch
Ken Jones
with Jeffrey Hobbs
tcl/tk
Practical Programming in
Tcl and Tk
FOURTH EDITION
The Tcl/Tk bestseller—fully updated for Tcl 8.4!
Authoritative coverage of every Tcl and Tk command in the core toolkits
State-of-the-art Tk GUI coverage for Tcl, Perl, Python, and Ruby developers
Covers all key Tcl 8.4 enhancements: VFS, internationalization and performance improvements, new widgets, and much more
Definitive coverage of TclHttpd web server—written by its creator
CD-ROM: Source and binary Tcl/Tk distributions and extensions for Windows, Linux, Solaris, and Macintosh, plus all code from the book
CD-ROM INCLUDED

Addison-Wesley, 2003

ISBN: 0-13-038560-3

$49.99 US

Tcl and Tk are scripting languages that have been around for more than a dozen years. This book's primary author, Brent B. Welch, was a student of Tcl's creator, John Ousterhout, and has worked with Tcl since its invention.

I've used Tcl almost since its inception, but I never really learned the Tcl/Tk hybrid. I read Ousterhout's *Tcl and the Tk Toolkit* about five years after it was published, and I found the information difficult to apply to current programs, because Tcl/Tk has changed over the years. This book is far more timely.

Practical Programming in Tcl and Tk, 4th Edition, is an impressive volume, at almost 900 pages. Both the table of contents and index are thorough, and the volume is extensively cross-referenced. Presenting so much material is difficult, but the cross-referencing allows a number of topics to be put on the back burner when they haven't been discussed. Furthermore, backward references are useful for providing a refresher on old topics.

One of the outstanding points is that this book spells out in which versions newer features were introduced. These tidbits are sprinkled throughout the text.

The book is hands-on, and you should try the examples as you're reading the text. All the listings are on the accompanying CD-ROM, along with distributions of Tcl, a number of extensions and a mirror of the wonderful Tcl Wiki as of April 2003. The CD-ROM is useful if you don't have a fast Internet connection. But what the book calls examples often are mere Tcl snippets. I would far prefer fewer, longer examples that could execute as complete programs.

The primary author is receptive to feedback and keeps an up-to-date set of errata on his Web site, which you should check. I'm anxious to apply what I've learned to modify a number of Tcl programs I use and to read other books on Tcl. I heartily recommend this book if you want to learn Tcl.

Archive Index Issue Table of Contents

Advanced search

From the Editor

Our Last Spam Issue?

Don Marti

Issue #121, May 2004

Fight spam now and prepare to eliminate it in the future.

Get a bunch of Linux professionals together these days and the topic inevitably turns to the spam problem. How much do you get, how many sneak through your filters and, of course, what are the bad things that happened to you when a spam filter decided to eat some important legitimate mail at the worst possible time.

If you're just getting your personal e-mail via POP or IMAP, spam might merely slow you down. But when you manage mail for a lot of users it's now a major cost. A flood of unsolicited bulk e-mail actually made an entire university's e-mail system obsolete. The bad news is that spam unfairly shifts the cost of marketing from senders to recipients. The good news is that Ludovic Marcotte's team made that cost as low as possible by deploying a reliable all-open-source mail system on Linux and commodity hardware. When you read the success story on page 44, notice that the site has planned to add more machines as the spam problem gets worse.

But there is hope for the spam problem to get better in the future. We won't go quite as ga-ga with “the end of spam is coming” predictions as some tech CTOs, but we can make spamming less lucrative for the perpetrators and maybe just another Net nuisance.

SPF fights forged spam by giving other sites' mail servers a way to check whether mail is really from you. Meng Weng Wong covered how to label your mail server as legit in our last issue, and now it's time to collect the benefits. Follow the steps in both articles and you'll quickly block out all the spam that claims to be from your own domain, then get more effective protection as more sites use SPF. If you're joining us in the middle, there's a link to the Web version of Part I.

System administration isn't all fun, glamour and beating back hordes of slavering spammers to the sounds of cheers and sighs of gratitude from spam-free users. So don't worry—we cover the important behind-the-scenes tools in this issue too. Now, time is only unidirectional in Stephen Hawking movies and reality. On your RPM-based Linux system, you can go back in time to correct a good upgrade gone bad. James Olin Oden describes how to do transactions and rollback with RPM on page 40.

If you want to get every last bit of the bandwidth you paid for, but not go over and hit steeper charges, check out the article by David Mandelstam and Nenad Corbic on page 54. Now you easily can make your Net traffic use whichever connection makes sense and use low-priced, small-business connections, such as DSL and cable modems, where you can.

Let's just take a moment to give thanks for the spammers while we have them. Think about it—we've learned to do groovy text classifying math, we've developed knowledge of the SMTP protocol and where future protocols can be better, and we've given users a new appreciation of where they would be without system administrators. Thanks, spammers.

Actually, no, on second thought, let's just put the parasites out of business. Enjoy the issue.

Don Marti is editor in chief of *Linux Journal*.

Archive Index Issue Table of Contents

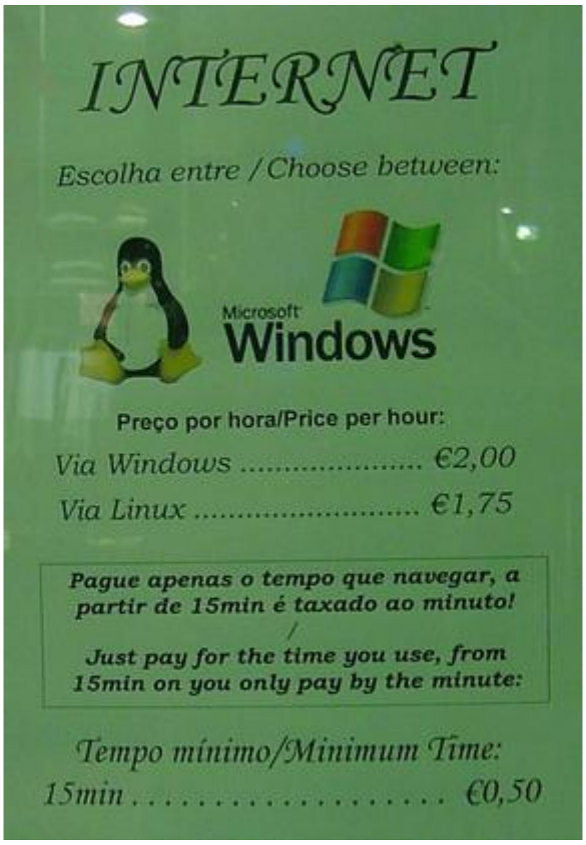
Advanced search

Letters

Readers sound off.

Use Linux and Save

I've been a Linux user and *Linux Journal* subscriber since the mid-1990s. I recently converted our home Internet machine to Linux after one too many virus attacks. Since the conversion, we've had zero problems with viruses and worms. Last week, my wife and I were in Funchal, Madeira, looking for an Internet café so we could e-mail our kids. The place we found had the attached ad in their window. Thought you'd get a chuckle out of it.



INTERNET

Escolha entre / Choose between:

Microsoft Windows

Preço por hora/Price per hour:

Via Windows €2,00

Via Linux €1,75

Pague apenas o tempo que navegar, a partir de 15min é taxado ao minuto! / Just pay for the time you use, from 15min on you only pay by the minute:

Tempo mínimo/Minimum Time:

15min €0,50

—

Frank

Advertiser Should Check Facts

The cartoon on page 27 [*advertisement for Hentzenwerke Publishing—Ed.*] bothered me because it is just so blatantly inaccurate. Opera, Firebird, WordPerfect and RealPlayer all are available for Windows, and some premiered on that platform. There are plenty of reasons why Linux is great without having to make up stuff!

—
Chris Schoenfeld

You're Supposed to *Eat* the Penguin?

We use this image for an evangelization campaign in Mexico (cofradia.org/modules.php?name=News&file=article&sid=7103). It is my son Luis Fernando eating a Tux toy.

—
Fernando Romo

Sewing Project

I have subscribed to your magazine for about a year and a half, and I love it. I have learned so much from it. I wanted to send you a picture of my son with a

Tux shirt my wife embroidered for him. Linux is very popular here and all my kids are getting an early start. Thanks.

—
Yossi 4Ever

Quality Grammar

I wanted to drop you a line to express my appreciation for the extra effort you folks appear to be making relative to the use of proper grammar. The absence of split infinitives and the correct positioning of the word only right next to the word or phrase it is intended to modify (rather than as early as possible in the sentence, a horribly bad practice that often thoroughly confuses the author's meaning) are just two examples of the high standard you are following. Thank you—some of us out here really do notice the difference.

—
craig.p

More Air Traffic Success

Thanks for the article on Linux in Air Traffic Control in the January 2004 *LJ*. Tom Brusehaver is quite right—Linux is ready for use in ATC. I'm just back from a tour of EuroControl, Europe's only multinational ATC. Our guide opened up the back of a controller's console so we could see the equipment running the displays, and when I asked him what their upgrade path was going to be from the old Compaq Alpha boxes that currently run the displays, he said that a pilot is now underway and Linux on x86 is scheduled to be deployed next year.

So, in 2005 Tux will be running the displays of the controllers responsible for airspace through which 19% of all European flights pass. That penguin's come a long way.

—
Jim Hague
Oxford, UK

Bad Webmaster, Bad, Bad!

I currently use Mozilla on my Linux desktop. My bank recently informed me that it will be supporting only Internet Explorer. Without an effective counterattack against critical IE-only Web sites, the Linux Desktop and the Mac are doomed to extinction.

—
William Mitchell

Check “Online Banking with Konqueror” for compatibility reports from smarter banks (home.in.tum.de/~strutyns/banking.php). —Ed.

Photo of the Month: Hey, Nice Outfit

Here is a picture of our little girl with Tux wearing her penguin Halloween costume.

—
Robert Henry

If we get you the material, will you make us one too? Photo of the Month gets you a one-year subscription. Photos to info@linuxjournal.com. —Ed.

The ESTABLISHED Rule

I enjoyed reading Mick Bauer's “Application Proxying with Zorp, Part I” in the March 2004 issue. In the “Firewall Refresher Course” section, Mr Bauer states that in a stateful packet filtering system “...firewall rules need address only the initiation of each allowed transaction.” He gives a comparative example between stateful and non-stateful systems in Tables 1 and 2. Although Mr Bauer was referring to stateful packet filtering systems in general, I'd like to point out that his explanation is not correct with respect to the Linux Netfilter package. My understanding of Netfilter, and the way I've been using it the last two years, is that a rule is required to permit packets that are part of an existing session to traverse the firewall (for example, `-m state --state ESTABLISHED`). This assumes that a default policy of DROP is used.

—
Terry Montgomery
President of Central Valley Area Linux Enthusiasts (cvale.org)

Mick Bauer's reply: I was generalizing. The point was to explain how stateful packet filtering differs from non-stateful filtering, not to explain how to configure Netfilter/iptables. Given the complexity of Zorp, the real topic of that article, I didn't want to spend more time than I had to on Netfilter basics. Anyhow, you're correct in pointing out that for the stateful magic to work in actual Netfilter/iptables practice, you need a single `-m state --state ESTABLISHED` rule at the start of each of your chains.

Internet Safety?

I am enjoying my second year as a subscriber to *Linux Journal*. *PC Magazine* had a really good issue this month concerning identity theft, spyware and viruses. This started me thinking. How does Linux combat these threats? Is it a concern within the Linux community? How does Linux stack up in this arena?

—
Robert Stewart

All of these problems are commonly spread through forged e-mail, and you can fight forged e-mail with SPF (page 50). If you don't run your own mail server, ask your ISP to use SPF to protect you. —Ed.

Erratum

Regarding “Manipulating OpenOffice.org Documents with Ruby”, *LJ*, March 2004—it's come to my attention that I incorrectly attributed REXML, the Ruby XML library, to Sean Chittenden. The actual author of that library is Sean Russell. —James Britt

UpFront

diff -u: What's New in Kernel Development

Zack Brown

Issue #121, May 2004

The **FAT filesystem** still is being maintained, and, in fact, **Frodo Looijaard** recently has added Linux support for some obscure FAT implementations that required unusual characters to delineate the directory index. One problem with FAT maintenance is the proliferation of implementations that should be supported under Linux. Another is that the MS-DOS version may not always clearly obey the official specification itself; in which case, as **H. Peter Anvin** has put it, the rule always should be to follow the WWDD rule (What Would DOS Do). The question of when to adhere to standards and when to slough them off in favor of something better is a perennial one in the free software world.

Michael A. Halcrow has begun work on an encrypted filesystem. According to his initial research, the best method will be to create an entirely seamless encryption layer above the root filesystem, with files marked as either encrypted or clear. Encryption keys associate with files as opposed to directories or block devices and so forth, but a directory can be tagged encrypted only, in which case all files within it are encrypted by default. Key and other metadata will be stored inside Extended Attributes. Standard file data, such as file size and permissions, will be encrypted, as far as is possible. Deleted files will be wiped from the disk as thoroughly as they can to reduce the possibility of recovery. A number of other interesting features appear to be in the offing as well.

Vojtech Pavlik has produced an FAQ for problems encountered with **Input drivers**. It covers such issues as how to get a list of Input drivers on the system, how to deal with mouse or keyboard misbehavior and many others. This is the first time these problems have been documented since the Input layer was rewritten during 2.5 development, and a lot of folks are overjoyed to have a central repository where their questions can be answered.

Ever since the **IDE** driver was rewritten, amid much turmoil and upheaval in the 2.5 development tree, efforts to fix its ability to be loaded as an external module have been put off until the rest of its code could stabilize. **Alan Cox** in particular rejected various patches from various developers attempting to make IDE loadable as a module. A recent attempt by **Witold Krecicki** seemed like a good attempt, but it looks like **Bartlomiej Zolnierkiewicz**'s patch is more likely to get into the 2.6 tree in the near future. Having worked intensively on the IDE driver during the 2.5 cycle, he's apparently been developing the loadable module patch for some time, and feels it adequately covers all the corner- and edge-cases.

It turns out that some of the **mailing lists** listed in the MAINTAINERS file as being the proper place to submit bug reports on various features actually require that users subscribe to the lists before posting their reports. This is not uncommon in the realm of mailing lists, but for Linux kernel development, it has been decided that all bug-receiving lists listed in the MAINTAINERS file must receive posts from anyone. This is the best way to welcome regular users to submit their bug reports. One of the main reasons to require subscription before posting has been the proliferation of spam and the attempt to keep it out of a development mailing list. Although there are various ways of dealing with that problem, it also has been decided that not all mailing lists in MAINTAINERS must be completely open, as long as they have a secondary list that does remain open to all bug reports.

A new port of the Linux kernel, called **Cooperative Linux**, has sprung up. Similar to User-Mode Linux (UML), Cooperative Linux is not a port to any particular hardware architecture but is intended to create a running Linux system on top of an existing system. Cooperative Linux already has run on an instance of **Windows 2000** and **Windows XP** successfully. The Cooperative Linux developers hope this will allow many users of other OSes to try Linux without the burden of having to replace their existing systems with complete Linux installations. Apparently, multiple instances of Cooperative Linux can run in parallel on a single system, and efforts toward running Cooperative Linux on top of Linux itself already are underway.

World-Record Distributed Chess Project

Carlos Justiniano

Issue #121, May 2004

On January 30, 2004, one of the world's top chess players, International Grandmaster Peter Nielsen, faced a distributed computing cluster called ChessBrain in an unprecedented man-vs-machine event. ChessBrain fought back as it navigated well-placed traps in a game that resulted in a draw.

The ChessBrain Project plays live chess games in real time, using distributed computing techniques similar to SETI@home and distributed.net. A central Linux-based server called a SuperNode coordinates thousands of distributed machines called PeerNodes.

Interest in the event was significantly greater than anticipated. As a result, a large number of PeerNodes descended on the SuperNode in a situation that resembled a denial-of-service attack. To address this potentially fatal situation, ChessBrain member Gavin Roy tweaked /proc filesystem entries to modify the TCP/IP stack's behavior. We also stopped the SuperNode from Denmark and modified the SuperNode source code to instruct remote PeerNodes to change the frequency of their connection attempts. We were able to address the situation in real time without rebooting a single machine.

ChessBrain's official Guinness World Record attempt was hosted by the Danish UNIX Users Group (DKUUG) at the Symbion Science Park in Copenhagen, Denmark, and supported by the US-based Distributed Computing Foundation. With a new record involving 2,070 computers from 56 different countries, ChessBrain has become the first distributed network to play a game against a single human opponent. This gives new meaning to “Powered by Linux!”

The ChessBrain team swaps notes with GM Peter Nielsen (right) after the game.

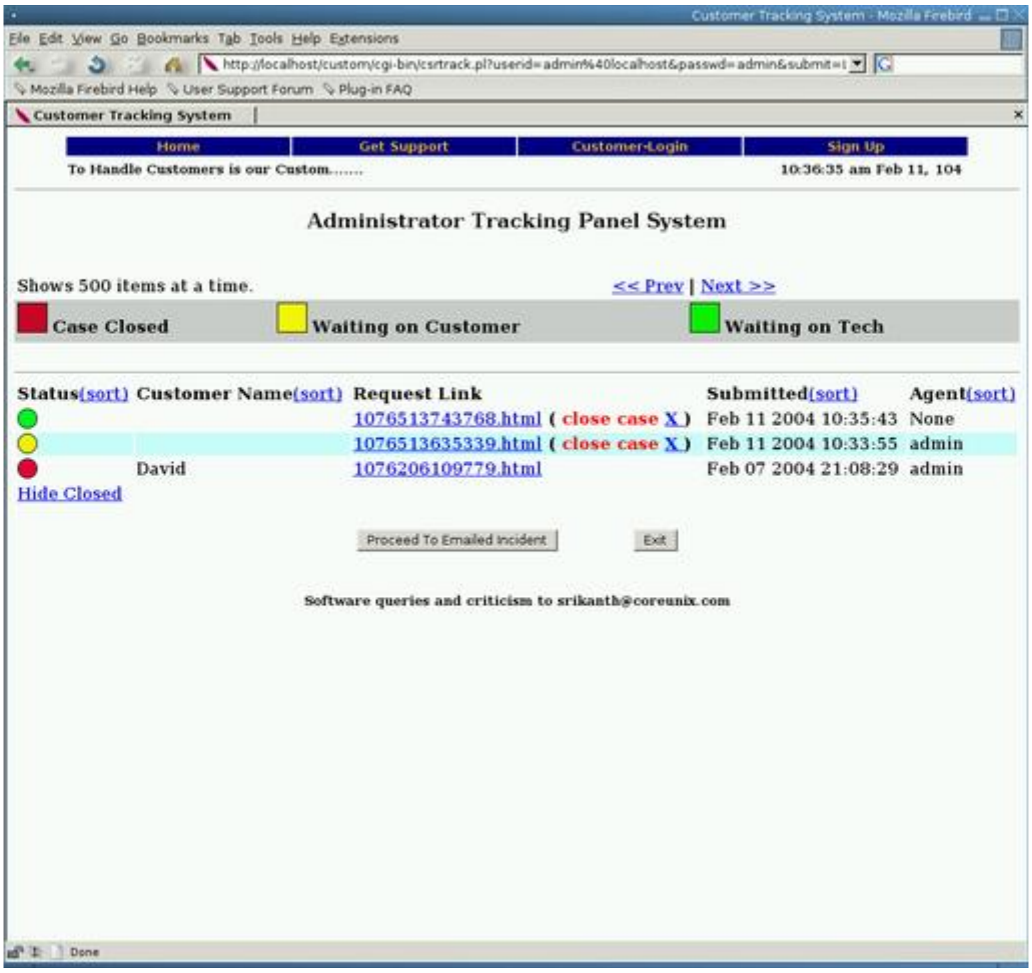
Carlos Justiniano is the founder of ChessBrain.net. His article “ChessBrain: a Linux-Based Distributed Computing Experiment” appeared in the September 2003 issue of *Linux Journal*.

Customer Request Tracking System: www.coreunix.com/custom

David A. Bandel

Issue #121, May 2004

Have you tried other ticket tracking systems and found that they're too complex or that they don't keep you and your customers in contact well enough? This ticket tracking system is simple to use for both customers and technicians, and it e-mails responses to customers so they know immediately when their request was processed. If you need a simple, easy-to-use tracking system, this might be the one for you. Requires: Web server; Perl; Perl modules CGI, DBI, Net::SMTP, Socket, File::Copy; and PostgreSQL.


Customer Tracking System - Mozilla Firebird
File Edit View Go Bookmarks Tab Tools Help Extensions
http://localhost/custom/cgi-bin/csrtrack.pl?userid=admin%40localhost&passwd=admin&submit=
Mozilla Firebird Help
User Support Forum
Plug-in FAQ
Customer Tracking System
Home
Get Support
Customer-Login
Sign Up
Proceed To Emailed Incident
Exit
Done

To Handle Customers is our Custom....... 10:36:35 am Feb 11, 104

Administrator Tracking Panel System

Shows 500 items at a time. << Prev | Next >>

🟥 Case Closed 🟨 Waiting on Customer 🟩 Waiting on Tech

Status(sort)	Customer Name(sort)	Request Link	Submitted(sort)	Agent(sort)
🟢		1076513743768.html (close case X)	Feb 11 2004 10:35:43	None
🟡		1076513635339.html (close case X)	Feb 11 2004 10:33:55	admin
🔴	David	1076206109779.html	Feb 07 2004 21:08:29	admin

Hide Closed

Software queries and criticism to srikanth@coreunix.com

LJ Index—May 2004

- 1. Percentage of surveyed enterprises that say they will increase Linux spending next year: 60
- 2. Percent surveyed that say they will increase Linux spending by 6% to 10%: 17
- 3. Percent surveyed that say they will increase Linux spending next year by 10% more than last year: 43
- 4. Percent surveyed that say their top spending on Linux will be in upgrading the OS: 35
- 5. Percent surveyed that say their top spending on Linux will go to server hardware upgrades: 30.5
- 6. Percent surveyed that say their top spending on Linux will go to data center migrations: 29.5
- 7. Projected increase factor in hardware spending at UAL Loyalty Services: 10
- 8. Percent surveyed that say data center migration is top priority among Linux-related projects for 2004: 30

- 9. Percentage of above who say they'll be spending up to $99,000 US on that migration: 13
- 10. Percentage of above who say they'll be spending between $100,000 and $499,000 US: 52.7
- 11. Number of months it takes IT to get an internal application to full deployment: 9
- 12. Thousands of PCs in a French study toward the possible replacement of Microsoft Windows with Linux: 17
- 13. Hundreds of applications in the same study: 6
- 14. Hundreds of servers in the same study: 4
- 15. Thousands of police computers switched to Linux from Windows in Lower Saxony, Germany: 11
- 16. Percentage of all European computers that now run Linux: 15
- 17. Percentage that said "Wait" in response to the *eWeek* survey of respondents' willingness to wait until 2006 for Microsoft's Longhorn OS: 35
- 18. Percentage that said "Switch to Linux desktop" in response to the *eWeek* survey of respondents' willingness to wait until 2006 for Microsoft's Longhorn OS: 40
- 19. Popularity position of Red Hat among all Linux distributions, according to Netcraft: 1
- 20. Fastest growth position of Debian among all Linux distributions, according to Netcraft: 1

- 1–10: SearchEnterpriseLinux.com
- 11: IDC
- 12–16: ZDnet
- 17, 18: *eWeek*
- 19, 20: Netcraft

packetbl: freshmeat.net/redir/packetbl/47435/url_tgz/packetbl-0.3.tar.gz

David A. Bandel

Issue #121, May 2004

This is a real-time blacklist module for iptables/Netfilter. If you run one or more mail servers, you can save processing time by running this module on your Linux router. The module can check incoming port 25 connections against the spam blacklists and drop those packets before your mail servers even see them. Requires: libdotconf and glibc.

They Said It

How are you going to write software that is useful for people if you don't know if or how they're using your features? Answer these questions: What percentage of bug reports to open-source projects are submitted by employees of financial services firms? Which industries have the highest patch submission to running copies ratio?

—Robert Lefkowitz, from his talk at OSCON

There's a fine line between participation and mockery.

—Scott Adams (www.photodude.com/index.shtml)

One way to fix this is to “increase the dimensionality of the discrimination hyperspace” (no, I am not making that phrase up).

—Bill Yerazunis, CRM114 & Mailfilter HOWTO

On the Web

Linux Users, Old and New

Heather Mead

Issue #121, May 2004

Learn how to update Bogofilter, deploy Linux desktops and clear up TCP/IP network traffic with this month's articles.

Back in November 2002, Nick Moffitt wrote a brief tutorial titled “Busting Spam with Bogofilter, Procmail and Mutt” (/article/6439) for the *Linux Journal* Web site. He provided a Bogofilter configuration that made it easy to mark incoming messages as spam or non-spam. Nick's article still receives a number of hits as people continue to look for ways to manage their mailboxes. The problem, however, is command-line switches to Bogofilter have been reversed as of March 2003, “so they now have the exact opposite effect”. In other words, follow the original article and you'll be training your spam filter to keep the spam and ditch the legit mail. To save readers some frustration, Nick wrote a follow-up article for our Web site, “Busting Spam with Bogofilter, Procmail and Mutt, Revisited” (/article/7436). If you used Nick's first implementation or are looking for another spam-fighting tool, be sure to read his update.

If you're in charge of a network whose servers use asymmetric TCP/IP routing, you may have noticed artificial bandwidth bottlenecks as all traffic goes out one interface, leaving the other idle. In “Overcoming Asymmetric Routing on Multi-Homed Servers” (www.linuxjournal.com/article/7291), Patrick McManus explains how to use source-based routing capabilities, similar to the ones used in high-end networking gear, to improve traffic flow in server environments. Specifically, he discusses the iproute2 package, which can be used to control routing behavior as well as to “set up interfaces, control arp behavior, do NAT and establish tunnels”. As Patrick states, “the key difference in an iproute2 world is the system may contain many different destination-based routing tables instead of a single global system table.”

When William Yu and Dominique Cimafranca were given the green light to install Linux on the desktops of a pilot group at their company, they were told the work had to be completed in half a day. Plus, if the pilot group didn't like the Linux desktops, their Windows desktops needed to be reinstated just as quickly. Given the age of the machines they were working with, plus memory limitations and the presence of a decent Ethernet infrastructure, they decided thin clients would be the best approach. Their article, “Desktop Guerilla Tactics: a Portable Thin Client Approach” (www.linuxjournal.com/article/7109), details their experience with using the VNC remote display system, assembling a floppy-based distribution and setting up a fat server. Read their article on-line to see how the pilot group reacted to the new desktop.

As more and more people make the move to Linux, whether it be at work or at home, interest in end-user applications grows. In response, the *Linux Journal* Web site has acquired some new columnists, including Dave Phillips, Chris DiBona and Bruce Byfield, who will write regularly on such topics as Linux audio, tasks for new Linux users and OpenOffice.org. Dave's Linux audio series already has a few columns posted providing introductions to AGNULA and Planet CCRMA. His March column, “At the Sounding Edge: OpenMusic and SuperCollider3” (www.linuxjournal.com/article/7432), discusses two new Mac audio application ports to Linux.

If there's a topic or application you'd like to see covered on the *Linux Journal* Web site, or if you'd like to write an article, drop me a line at info@linuxjournal.com.

Heather Mead is senior editor of *Linux Journal*.

Best of Technical Support

Our experts answer your technical questions.

Slow Backups

I moved our Samba server (Red Hat 7.3, PII) to a new PC (Red Hat 9, P4). I have a cron job set up to create daily backups from shares using smbtar. I have installed all the latest patches using up2date. Problem: this backup script is running much more slowly on the new configuration than on the old one. Any ideas why this might be?

—
Zoltan Sutto

sutto.zoltan@rutinsoft.hu

My first guess is Ethernet drivers. Make sure they are the latest and greatest. I also have had issues with Ethernet auto-negotiating speed. Make sure you are at 100BT/full duplex.

—
Christopher Wingert

cwingert@qualcomm.com

If you really wanted to analyze the problem, you'd start by running the smbtar script with tracing turned on (the -x option to bash). That's because smbtar is a shell script. Then, you could eyeball it to see which commands were taking a long time. You also could (more invasively) edit a copy of the script, inserting calls to take timestamps (relative and absolute) between calls to external commands. These could be written to a profiling file or simply sent to the system logs using the logger command. You can use shell expressions like:

```
START_TIME="$(date +%s)"; REL_TIME="$START_TIME"
REL_TIME="$(( $(date +%s) - $REL_TIME ))"
...
```

to get the current time (as a number of seconds since the epoch in 1970). Thus, the total elapsed time for your script would be the current time minus the $START_TIME that you set as the first line of the script.

Also consider that differences in your configuration might be introducing some odd network name services delays, for example, if your old /etc/hosts file had some entries that made reverse DNS queries work and the new installation has failed to preserve those, or if your old /etc/nsswitch.conf was only checking local files and your new one is somehow querying NIS, LDAP or winbind (MS Windows domain) sources. Because winbind is in newer Red Hat systems after 7.3, it could be the culprit.

Performance tuning is a process of taking measurements (profiling) to find bottlenecks (analysis) and eliminate those where possible (tuning). Usually the elimination of bottlenecks involves finding cases where the system is doing work unnecessary to your application, for example, querying network-based directory services rather than simply using local files.

Sometimes you should consider an entirely different approach to the task at hand. In this case, I'd seriously consider not using smbtar to back up these Samba shares. You simply can use rsync to synchronize the selected (shared) directory trees to one large holding disk on the system with the tape drive. Then, back that up directly to tape.

—
Jim Dennis

jimd@starshine.org

It could be that your new system is not getting as much throughput to your hard disks as it should be. I'm assuming you have IDE disks. Default installs on some Linux distributions don't necessarily enable DMA by default; it has to be enabled explicitly after install. You can use hdparm to verify/test your drive (in my case, my system is on /dev/hda):

```
[root@hamtop ~]# hdparm /dev/hda

/dev/hda:
multcount    = 16 (on)
IO_support   =  0 (default 16-bit)
unmaskirq    =  0 (off)
using_dma    =  1 (on)
keepsettings =  0 (off)
readonly     =  0 (off)
readahead    =  8 (on)
geometry     = 3648/255/63, sectors = 58605120, start = 0
```

Check the using_dma entry. If yours is set to 0, that could explain it. Try setting it to `hdparm -d1 /dev/hd`*X*, where *X* is your drive letter. Then test it:

```
[root@hamtop ~]# hdparm -tT /dev/hda

/dev/hda:
Timing buffer-cache reads:   128 MB in  0.82 seconds = 156.10 MB/sec
Timing buffered disk reads:  64 MB in  2.68 seconds = 23.88 MB/sec
```

You should see the buffered disk reads go up considerably compared to what you get from running the same test without DMA enabled. Thoroughly test the drive with DMA enabled before relying on it, as in rare cases older drives don't behave well with this set. If this does fix it, read up on how your particular distribution can be made to enable this at boot. In the case of Red Hat, it can be controlled through /etc/sysconfig/harddisks.

—
Timothy Hamlin

thamlin@zeus.nmt.edu

How to Recover a Kernel .config File?

I have reconfigured the Linux kernel on my computer to version 2.4.22, but at the boot screen, I still have the option to choose between version 2.4.20-8 and 2.4.22. My problem is I do not have the .config file for the 2.4.20-8 kernel, and I'd like to know whether there is a command to generate this file?

—
Jan Nicolas Myklebust

jan-nicolas.myklebust@cnes.fr

If this is the default Red Hat kernel, you can unpack the kernel source package and grab the .config file from the /usr/src/linux-2.4/configs directory.

—
Christopher Wingert

cwingert@qualcomm.com

There isn't a command to generate a .config file from a kernel image in 2.4.x and earlier. In the new 2.6 kernels, a compile-time option supports this.

—
Jim Dennis

jimd@starshine.org

bash without History

The February 2004 BTS column had a question about hiding mistakenly entered information from the bash history. If you kill your own bash process with `kill -9 $$` instead of logging out, it doesn't write history to disk.

—
Jack Coates

jack@monkeynoodle.org

Can't Make a Partition on Free Disk Space

The current partitioning on my Red Hat 9 system is:

```
hda1 20GB Windows
hda2  7GB Linux /
hda3 12GB Linux /usr
swap  1GB
```

I have resized hda1 down to 8GB using GNU parted, thus getting 12GB of free space. Now I want to make a new Linux partition on the unused 12GB. The problem is, the parted `mkpart` command simply says `can't make partition` and the `fdisk n` command says `delete a partition before you make new partition`.

—
Hiroshi Iwatani

HGA03630@nifty.ne.jp

Sounds like you have four primary partitions already, and the maximum is four. You need to delete a partition and add a logical partition, which can encompass many more partitions. I would turn off swap, delete the swap partition, add a logical partition including all free space, add a new swap partition, run `mkswap`, add and format your data partition and then turn on swap. You should also update /etc/fstab for the new swap and data partition.

—

Christopher Wingert

cwingert@qualcomm.com

Quick Crossover Networking

How can I use a cross-link Ethernet cable to transfer data from one computer to the other when both are Debian sarge and when one is sarge and the other is Microsoft Windows?

—
Akos Zelei

azelei@freemail.hu

You simply can give each of the two machines any arbitrary IP addresses from the same network (I'd recommend using the RFC1918 address blocks reserved for these purposes: 192.168.x.*—so call one 192.168.1.1 and the other 1982.168.1.2). If you choose the addresses wisely (or follow my example) you can leave the subnet and broadcast values at their defaults. You then should be able to ping each from the other. At that point, you also should be able to run any normal TCP/IP protocols over that link. You can use the IP addresses or add entries for left and right in the /etc/hosts files on each. At that point you'd use rsync, scp or any protocol you liked across them. As for the Windows system: you can create a static IP address configuration manually and either use its native file sharing (configure Samba on the Debian GNU/Linux system) or install the Cygwin for MS Windows suite and use rsync over SSH and so on.

—
Jim Dennis

jimd@starshine.org

If you don't want to set up the Linux system as a Samba server, put putty on the Windows box (www.chiark.greenend.org.uk/~sgtatham/putty). Or, if the Windows box is already set up to share files, you can use smbclient from Linux.

—
Don Marti

info@linuxjournal.com

New Products

Altix 350, IVR100B, Desktop/LX Pocket PC Edition and more.

Altix 350

SGI's Altix 350 server scales from one to 16 64-bit Itanium 2 processors (regular as well as low voltage) and up to 192GB of global shared memory in a single system. The 350 also uses the 6.4GB/sec SGI NUMAlink interconnect. It is capable of independently scaling across processors, shared memory and/or I/O on a single, standard chassis with different expansion modules, making it suitable for demanding technical applications. Along with the 350, SGI offers ProPack software, which includes tools, libraries and performance improvements that build on system, data and resource management features in the standard Linux distribution.

Silicon Graphics, Inc., 1500 Crittenden Lane, Mountain View, California 94043, 650-960-1980, www.sgi.com.

IVR100B

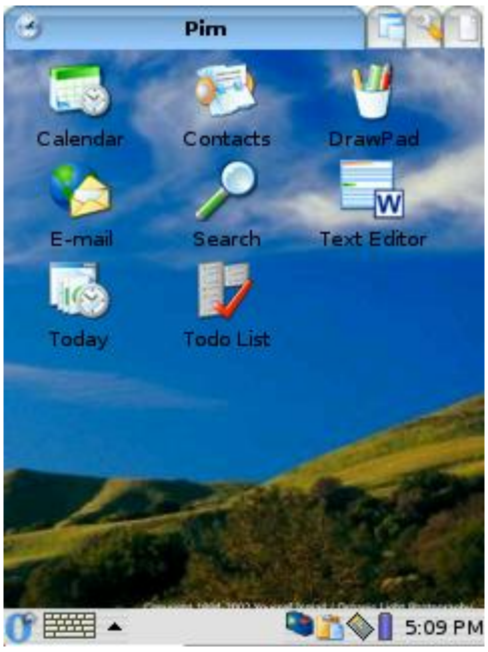
The IVR100B is a rackmountable telecom application server for interactive voice response (IVR) applications, based on the GNU/Linux OS and Bayonne. It ships standard with a four-port Pike Inline GT voice resource card that is expandable to 24 ports. The IVR100B features a Pentium-class SBC with at least 32MB of RAM, an onboard USB and 10/100 Ethernet and a Quantum Fireball LM IDE hard drive. A standard set of IVR applications capable of interfacing with database, Web and mail servers are included with the IVR100B.

Open Source Telecom Corporation, 278 Hope Street Suite E, Mountain View, California 94041, 866-688-6423, www.ostel.com.

Desktop/LX Pocket PC Edition

Lycoris announced the Desktop/LX Pocket PC Edition of its Linux OS customized for handheld devices. Based entirely on open standards, this edition supports wired, USB, infrared and 802.11b networking. It also provides a full-service PIM suite, supports full HTML and CSS 4 protocols and POP3 e-mail and enables playback of audio, video and streaming-media formats. Device input support includes gesture-based handwriting recognition, onscreen keyboard, built-in touchscreen, pickboard and a physical keyboard. Desktop/LX Pocket PC Edition also offers support for select StrongARM- and XScale-based processors and chipsets.

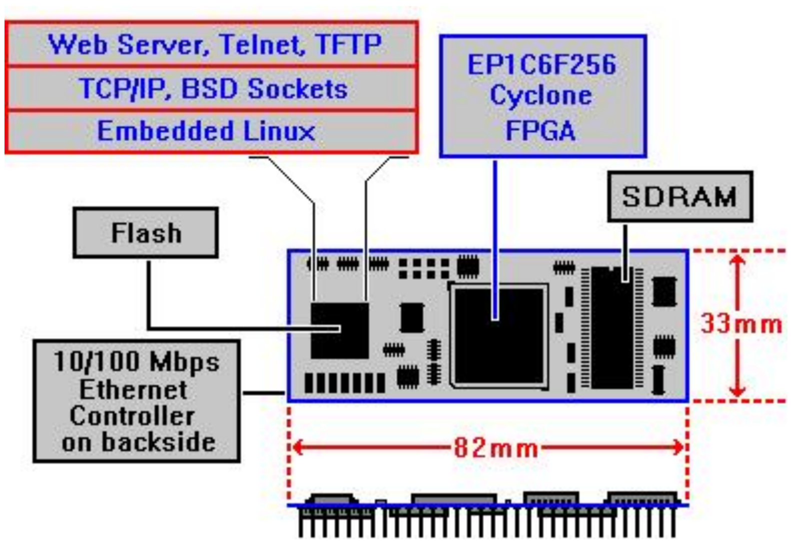
Lycoris, 26828 Maple Valley Highway #259, Maple Valley, Washington 98038, 425-738-6604, www.lycoris.com.



ADNP/ESC1 with uClinux

SSV Embedded Systems releases the ADNP/ESC1, an FPGA-based DIL/NetPC built specifically for embedded softcore computing (ESC). An Altera EP1C6F256 Cyclone FPGA is used in the ADNP/ESC1 instead of an MCU. The ADNP/ESC1 offers a 32-bit NIOS-Softcore processor with two UARTs, 20-bit PIO, SPI, JTAG, an IDE CompactFlash interface and a 16-bit expansion bus with chip select outputs and interrupt inputs. The module includes 12MB of SDRAM, 8MB of Flash and a 10/100 Ethernet controller. A starter kit is available for system integration. It includes a networking prototyping board, sample applications and the uClinux OS, based on the 2.4 kernel.

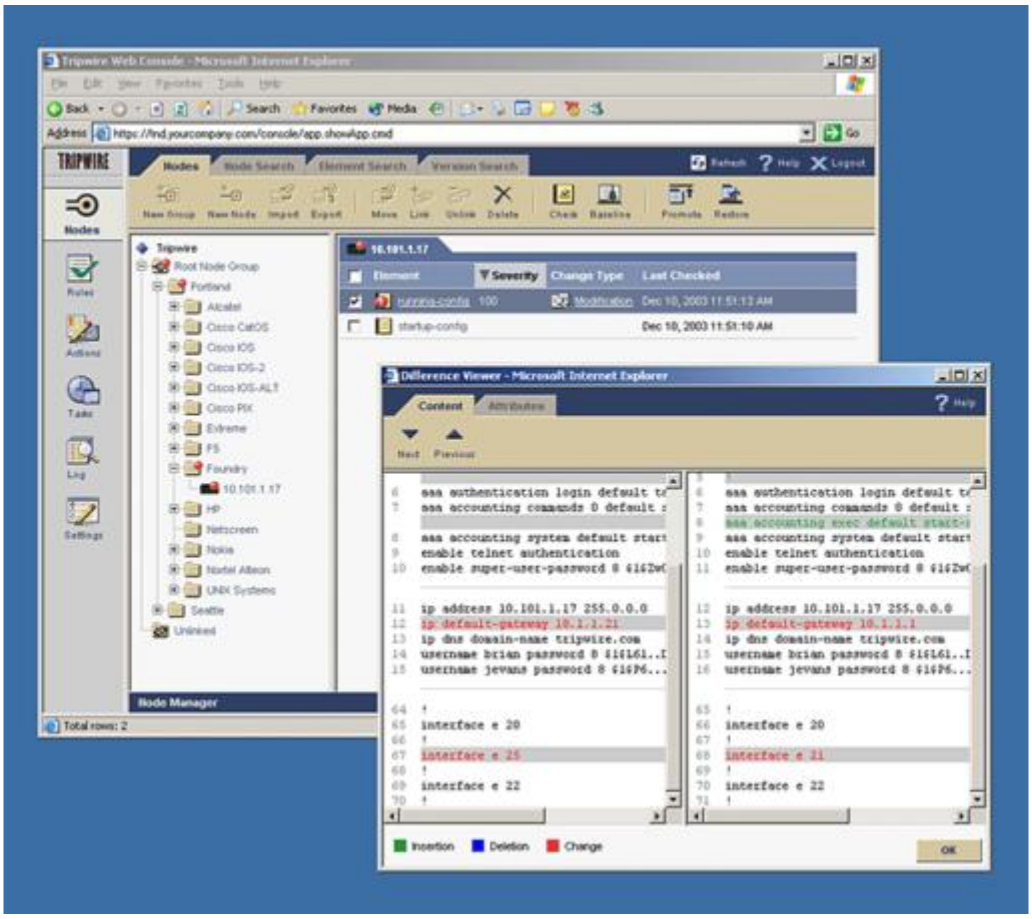
SSV Embedded Systems, Heisterbergallee 72, D-30453 Hannover, Germany, www.ssv-embedded.de.



Tripwire for Network Devices 3.0

Tripwire for Network Devices 3.0 is multivendor network configuration management software that centrally manages, monitors and reports changes made to network components. In addition to heterogeneous device support, Tripwire offers version control, in which an archive of configurations for every device is maintained and updated automatically whenever change is detected. Tripwire for Network Devices can scale to tens of thousands of nodes that can be organized in logical groups. It integrates with user authentication, access and accounting applications to manage passwords and user access rights. Tripwire also offers baseline restorations, real-time integrity scans and proof of conformance.

Tripwire, Inc., 326 SW Broadway, 3rd Floor, Portland, Oregon 97205, 800-874-7947, www.tripwire.com.



10.101.1.17

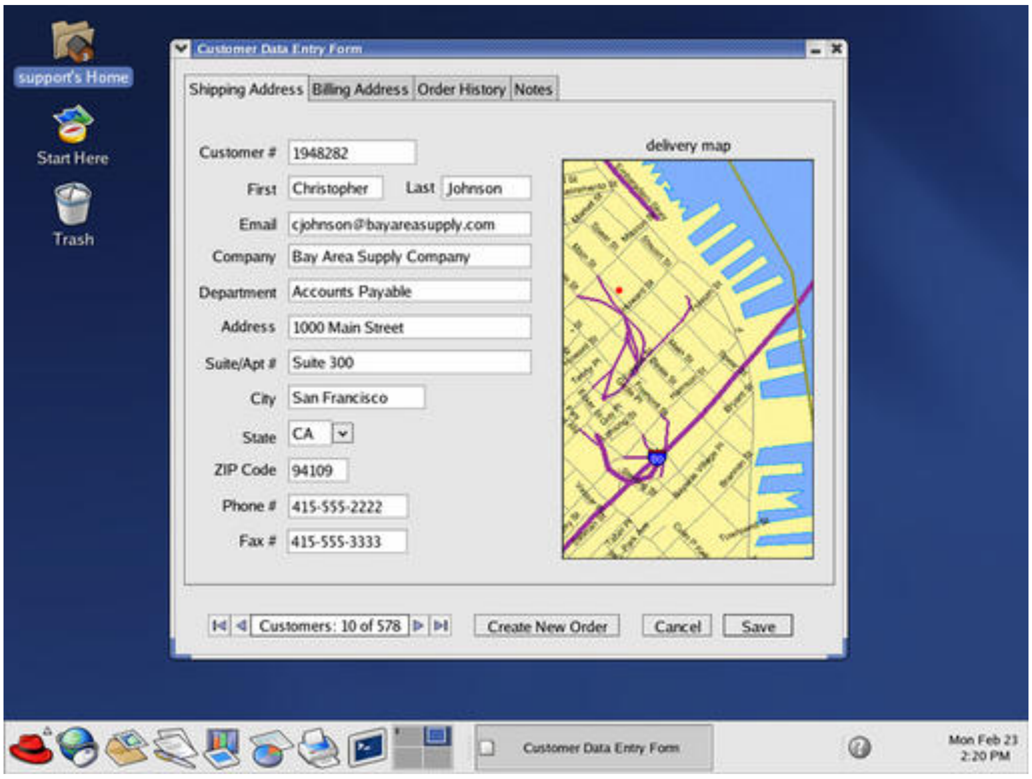
Element	Severity	Change Type	Last Checked
running-config	100	Modification	Dec 10, 2003 11:51:13 AM
startup-config			Dec 10, 2003 11:51:10 AM

```
aaa authentication login default t   aaa authentication login default t
aaa accounting commands 0 default    aaa accounting commands 0 default
                                     aaa accounting exec default start-
aaa accounting system default start  aaa accounting system default start
enable telnet authentication         enable telnet authentication
enable super-user-password 8 $1$Zw(  enable super-user-password 8 $1$Zw(

ip address 10.101.1.17 255.0.0.0     ip address 10.101.1.17 255.0.0.0
ip default-gateway 10.1.1.21         ip default-gateway 10.1.1.1
ip dns domain-name tripwire.com      ip dns domain-name tripwire.com
username brian password 8 $1$L61..I  username brian password 8 $1$L61..I
username jevans password 8 $1$P6...  username jevans password 8 $1$P6...

!                                    !
interface e 20                       interface e 20
!                                    !
interface e 25                       interface e 21
!                                    !
interface e 22                       interface e 22
!                                    !
```

REALbasic 5.5

REALbasic 5.5 is a development tool for creating cross-platform software for Linux, Windows and Mac. REALbasic includes the VB Project Converter tool to migrate tables, forms and code to REALbasic to get applications ported to Linux or Macintosh. REALbasic 5.5 supports Linux for x86 Intel platforms running Red Hat Enterprise or SuSE, as well as other distributions with the GTK+ 2.0 and CUPS libraries. Remote debugging is included so Linux applications can be tested and debugged from either Windows or Mac environments. Other additions and upgrades for version 5.5 include improved user interfaces, improved MS Office compatability, extended Mac OS X support, better database support and support for SOAP, XML and APIs.

REAL Software, 1705 South Capital of Texas Highway, Suite 310, Austin, Texas 78746, 512-328-7325, www.realsoftware.com.



Customer #	1948282
First	Christopher
Last	Johnson
Email	cjohnson@bayareasupply.com
Company	Bay Area Supply Company
Department	Accounts Payable
Address	1000 Main Street
Suite/Apt #	Suite 300
City	San Francisco
State	CA
ZIP Code	94109
Phone #	415-555-2222
Fax #	415-555-3333