Linux Journal Issue #103/November 2002

Features

Bridging the Digital Divide in South Africa *by Linda Martindale*
When you have to localize both Mozilla and OpenOffice, do you have to teach your translators two sets of tools? No—just use KDE's KBabel.

Radio E-Mail in West Africa *by Wayne Marshall*
To the users, it looks like regular e-mail. But behind the scenes, a Linux-based project is using HF radio to move it hundreds of kilometers without a wire or even a repeater.

Introduction to Internationalization Programming *by Olexiy Ye Tykhomyrov*
When your software gets new users who prefer a different language, what are you going to do? Learn the fundamentals of POSIX locales and GNU gettext now, so that you can make your program multilingual later.

Indian Language Solutions for GNU/Linux *by Frederick Noronha*
Some of the hardest languages to support are also some of the most widely spoken. Here's an overview of the projects to make Linux work with the two languages on our cover and more.

Indepth

Playing with ptrace, Part I *by Pradeep Padala*
You might have used strace to see what system calls a program makes. strace strace and you'll see it uses the ptrace call. What's that? Here's what.

Bridging the Digital Divide in South Africa

Linda Martindale

Issue #103, November 2002

The Open Source community addresses the sensitive language issue in South Africa.

South Africa has 11 official languages. It also has one of the greatest divides between rich and poor in the world, and this divide is most evident in the technology area. The IT world unwittingly has excluded the masses as technology has raced on leaving many South Africans behind. Socio-economic circumstances, imbalanced education policies under the apartheid regime, as well as language barriers, are some of the factors recognised in this exclusion.

Dr Neville Alexander, political activist, who was imprisoned on Robben Island during ex-president Nelson Mandela's time there, and director of the Project for the Study of Alternative Education in South Africa, has been a key player in the language debate. He emphasises the need to move away from being a monolinguistic society:

> An English-only, or even an English-mainly, policy necessarily condemns most people, and thus the country as a whole, to a permanent state of mediocrity, since people are unable to be spontaneous, creative and self-confident if they cannot use their first language.

Navigating the cyber world is daunting enough for first-time travelers, without having to do it in a language that is not their own. The translation of computer programmes into South African languages such as Xhosa was virtually uncharted territory until last year. Most computer software is only available in English and poorly supported in South Africa's second language, Afrikaans. The other nine official languages are not visible in any form of software—interesting when considering that Zulu is the most commonly spoken language in South Africa, with Xhosa a close second.

To address this, translate.org.za (www.translate.org.za) has been set up to translate computer software into various official languages of South Africa. This is the first project of The Zuza Software Foundation, a nonprofit organisation aiming to promote development and open-source software in Africa. Zuza, meaning “gift given freely” in a local African language, is working on proposals directed at internationalisation, business development and education using open-source software. Zuza is responsible for another project, Linuxlab (www.linuxlab.org.za), which provides disadvantaged schools with refurbished computers and free support.

The translation project began with KDE, but the focus has changed due to the fact that it is not cross-platform. KDE 3.0 includes Xhosa, Zulu and Venda. The translation and technical teams are now focusing on Mozilla and OpenOffice because they run on Windows, Linux and Mac, making it easier for any computer user to install it and exposing a greater user base to open source. Zuza unashamedly uses these products to promote OSS.

The translation is done using gettext, the GNU localisation architecture and tools. The advantage of gettext is that it separates programming from translating—translations can be added or improved because they are not compiled into the application. The team is using KBabel to do the actual translation. A GUI tool is vital, as the translators are not highly technical, and it is an easier environment in which for them to work. Mozilla and OpenOffice have their own format of translatable text, which is very different from gettext PO files.

Founder and Director of translate.org.za Dwayne Bailey says one tool is better than three:

> We have invested a lot in learning KBabel, which works on gettext PO files, and expecting our team of translators to learn a new tool or to edit files manually is not productive. We don't want an environment where they need three tools, but would rather allow all of our work to be used seamlessly across all three projects.

In order to address this, the team built scripts to convert Mozilla DTD files to PO files and similarly, use scripts to convert OpenOffice to PO and back. These shield the translators from the different formats and allow the teams to reuse existing work across all projects easily. And, the conversion of these formats is in the hands of technically skilled people and not the translators.

Internationalisation in Linux with KDE is well advanced but as new languages are added, shortcomings are highlighted, and this brings new features to the internationalisation framework. In projects like Mozilla and OpenOffice, you can

observe a certain internationalisation immaturity in that they don't adequately cater for different plural forms, something that KDE is managing well. However, one of the advantages of open source is that these shortcomings are addressed quickly. Another area that would help develop the project and assist internationalisation would be the creation of a web-based translation tool to allow remote translators to contribute to the project easily without having to install software or even use Linux.

Bailey clarifies why he puts so many hours into the translation project:

> The Free Software community conveys a spirit of cooperation and sharing, and after you have worked in the environment for a while and realise how much you have gained, you often feel compelled to contribute something back—even if it is just a well-thought-out bug report. And deep down, there is probably a feeling of some fame. The bug reports got boring after a while, and localisation was an area I felt I could contribute to without major programming expertise.

It is not only about the technical side of the project—language is a highly sensitive issue in South Africa. Neville Alexander states "...language policy and practice in our post-apartheid society is a critical component of the ensemble of antiracism strategies on which we depend for the real and visible transformation of this country."

No commercial software vendors have addressed adequately the language issue in South Africa, but in one year the Open Source community has. Bailey explains,

> In South Africa many languages have been marginalized through the history of apartheid, which has led to a lack of language pride. Seeing Linux users working in German and French environments made me realise that this could do the same for South African languages. I hope that simply allowing people to use the computer in their mother tongue will stimulate pride in their language. Also, learning something in your mother tongue is naturally easier.

South African Linux support and development company, Obsidian Systems, developed and sponsored this project, sharing their premises, resources and expertise to get the translation up and running. The Shuttleworth Foundation, recently publicised due to its founder's trip to space as the "First African in Space", has provided the finances for this project that has just begun to scratch the surface of endless opportunity in computer accessibility. Mark Shuttleworth, who sold Thawte to VeriSign for $575 million, developed his product on open-source software. One of his foundation's key aims is to

develop open-source software in South Africa, which is beginning to gain inroads into the lifeblood of the mainstream IT sector, both governmental and private.

The translation team is targeting the 11 official languages of South Africa first. Once those projects are mature, the team will look beyond their borders to other parts of the African continent. “Open source provides a way for Africans to help themselves—not to have to wait for the first world—but to get up and do it for themselves! Nobody else is going to translate software into Swahili”, says Bailey.

At this stage translate.org.za is also providing employment for young university graduates who are the translators, as well as students who are working off their fees owed to the University of Cape Town through bursaries.

Obviously the issues that surround language, inequality and poverty are broad and impact every sphere of life, but Zuza aims to play their part. “Translation does not remove all barriers to computer access”, says Bailey, “but it helps to eliminate one. This together with low-cost computers, open-source software and low-cost internet access will go a long way to making a dramatic IT impact on South Africans, especially the disadvantaged.”

The first step is getting software translated, but without actually getting people in front of computers, the translation would not have the intended impact. This is where a relationship with the second arm of Zuza, Linuxlab, is vital. The two projects need to work hand in hand. Linuxlab is looking at low-cost, thin-client solutions for disadvantaged schools. In South Africa these schools often serve people whose first language is not English.

Linuxlab's mission is to promote universal access to computers and Internet in schools, allow educators to create and refine open content and empower learners with a high-quality, high-tech IT learning environment. The project's director, Dr Evan Summers, believes that the culture of learning and sharing of knowledge in the scientific tradition is furthered by the GNU Free Software Foundation, which respects learners as participants in the software community, encourages unhindered enthusiasm for information technology and fosters a culture of collaboration in our information society.

The first Linuxlab to be deployed was the Alexander Sinton School in Cape Town, using a Linux server sponsored by The Shuttleworth Foundation and recycled 486 PCs as diskless X terminals with the goal of achieving a cost-per-seat of under $100 including network and server.

Mexico's Red Escolar (Schools Network), a similar concept, failed due to the way the programme was implemented. Dr Edgar Villanueva Nunez states in a widely publicised rebuttal to the general manager of Microsoft, Peru:

> ... the driving forces behind the Mexican project used license costs as their main argument, instead of other reasons specified in our project, which are far more essential. Because of the conceptual mistake, and as a result of the lack of effective support from the SEP (Secretary of State for Public Education), the assumption was made that to implant free software in schools it would be enough to drop their software budget and send them a CD-ROM with GNU/Linux instead. Of course this failed, and it couldn't have been otherwise, just as school laboratories fail when they use proprietary software and have no budget for implementation and maintenance.

In this attempt to introduce the masses to computer technology, it is vital that high value is placed on skills transfer. With this in mind, Dr Summers conducts regular free Linux workshops for educators and has developed Precis, a new Java-inspired programming environment suitable for teaching Linux programming in schools. The schools drive the project with involvement from both educators and learners, with Linuxlab helping to get the initial “seed” lab up and running, transferring skills and offering ongoing support to the school. What Linuxlab asks in return is that these schools co-opt educators from neighbouring schools to participate in the Linux workshops and that they in turn assist those schools in setting up their own labs when they are e-ready. And so the process continues.

The goal to have a high-quality IT learning environment available to all learners is not a mission impossible. Summers explains:

> An interesting trend is that organisations in Europe and the US are looking to offload large quantities of old PCs that are not suitably upgradeable for Windows XP, since dumping them would contravene US environmental legislation owing to their lead content. It is predicted that the number of PCs that will become obsolete in the US alone over the next ten years is in the hundreds of millions. This could give further impetus to the PC refurbishing industry in South Africa, creating welcome employment opportunities and increasing the availability of inexpensive low-spec refurbished computers, which are suitable as diskless Linux X terminals for school labs.

Linuxlab assists schools in identifying potential sources for old PCs that can be recycled as diskless X terminals. At Alexander Sinton School the server uses the Linux Terminal Server Project and the KDE desktop, including the localisation

from translate.org.za. Volunteers from the Cape Town Linux community deploy the labs. The school recruits members from the community and neighbouring schools to assist with preparing the lab, including the network cabling and furniture, in order to keep the costs of the seed lab under $1,000. This empowers educators in disadvantaged schools to realise the dream of providing a high-quality IT learning environment for their learners.

The gap between those empowered by technology and those who have been excluded must not widen. The only thing that needs to be broadened is the thinking of those who believe technology is for the privileged few. The Open Source philosophy flouts this belief. Translate.org.za and Linuxlab.org.za are two projects that are striving to make technology accessible to all South Africans.

email: linda@obsidian.co.za

Linda Martindale is a freelance journalist working from Cape Town, South Africa with a great interest in open-source software and Linux. She spends much of her time on local social issues and has recently published her first book, Celebrate Hope, which is a collection of success stories coming out of the disadvantaged areas of the city.

Archive Index Issue Table of Contents

Advanced search

Radio E-Mail in West Africa

Wayne Marshall

Issue #103, November 2002

Remote networking with high-frequency (HF) radio and Dan Bernstein's qmail.

Editors' Note: The complete version of this article, was later published on the Linux Journal web site.

Deep inside the warm green interior of Guinea, centered in the frontal lobe of West Africa, international rescue workers in the widely scattered towns of Dabola, Kissidougou and Nzérékoré now enjoy regular internet e-mail, delivered straight to their own desktops. There isn't a telephone line or satellite dish in sight. Instead we are moving the mail over distances of hundreds of miles—over jungled mountains and high palmy savannahs—using high-frequency (HF) radio. Our project is called Radio E-mail, and here is its story.

The Republic of Guinea is a cashew-shaped nation on the Atlantic, ten degrees north of the equator in West Africa. It is a beautiful and resource-rich nation, about the size of Oregon. As far as African countries go, Guinea is a calm pocket of peace and stability and generally doesn't attract a lot of attention. But Guinea quietly has played a heroic role in the theater of world events in recent years, providing a safe and welcome refuge for as many as half a million people displaced by brutal wars and civil upheavals in the neighboring countries of Sierra Leone and Liberia.

The International Rescue Committee (IRC) has one of their largest operations in Guinea, providing services and support to a population of up to 200,000 refugees in many camps established throughout the country. I became involved with IRC when my wife accepted the position of country director for the program in the summer of 2001. Soon we were traveling on an inspection tour of the camps, making the long road trip to visit the program's three field offices “up-country”.

Traveling outside the capital city of Conakry, one immediately finds that Guinea has little infrastructure, especially in the way of electrical grid and telecommunication systems—to say nothing of broadband access to the Internet. So IRC field offices must provide their own infrastructure: diesel generators for electricity and HF radio sets to communicate with other offices and mobile units, which can be up to hundreds of miles apart.

Expecting this isolation and general lack of connectivity, I was quite astonished to find a radio operator using his equipment to make a binary file transfer from his desktop PC to another field office—wirelessly! On top of the operator's radio set, connected to the serial port of his PC, sat a dingy black box labeled "9002 HF Data Modem". The operator used a proprietary, MS-DOS-based program to make his file transfers, but I immediately began wondering. If this device is moving binary data over the ether of radio, why couldn't we set it up with Linux and network with PPP connections as well?

Since IRC owned most of the equipment already and because we would be using Linux and other freely available software, the system could be implemented at negligible cost. I developed a design and specification for the system, and the project we call Radio E-mail has been continuously operational since January 2002.

HF Goes the Distance

If you have been making the move to wireless lately, most likely you are working with the microwave, high-bandwidth frequencies of 802.11b. If so, you know that on a clear day you *maybe* can get a line of sight connection out ten miles or so. HF radio is another animal. Its longer waves reflect off the ionosphere to follow the curvature of the earth, giving HF signals a range in the hundreds of miles. From Conakry to Nzérékoré (IRC Guinea's most distant field office), HF easily covers a straight-line distance of over 375 miles (600 kilometers).

So the great advantage of HF is it can go the distance, leaping the obstacles in its path with aplomb. Now for the bad news: where HF wins the wireless game in range, it loses its pants in data capacity. If 802.11b is considered broadband, think of HF as slim-to-none-band. The radio modems we are using here are spec'd at an anorexic 2,400 baud!

And wait, it gets worse. Two-way radio is the classic half-duplex medium of communication. That is, you are either transmitting (push to talk) or receiving, not both at the same time. This, plus the robust error-checking protocols implemented by the modem hardware, means the actual link experience is more on the order of 300 baud. Does anyone remember 300 baud? Unless you

measure your patience with radio-carbon, your dreams of remote login sessions will be dashed and splattered.

However, for classic store-and-forward applications like text-based e-mail, the bandwidth limitation of HF radio is workable. We do need to pay close attention to our configuration and try to optimize as much as possible. With HF radio, every packet is precious.

Radio Network Topology

Given these capabilities and limitations of HF, our design strategy for the project uses radio modems in a topology of field offices, as shown in the reference network of Figure 1. In Conakry we have a full-time internet gateway on the host coyah, and the radio modem on the host congo serves as the dial-in hub for each of the three field offices. We establish periodic PPP links between the field and Conakry and use our choice of carefully selected client/server protocols over TCP/IP. Although we might have implemented e-mail to the field using UUCP instead, our TCP/IP system is easier to configure and integrate with the existing network and e-mail system. This setup also is converted readily to use faster telecommunications linkups, such as land lines or satellites, if and when these become available in the field.

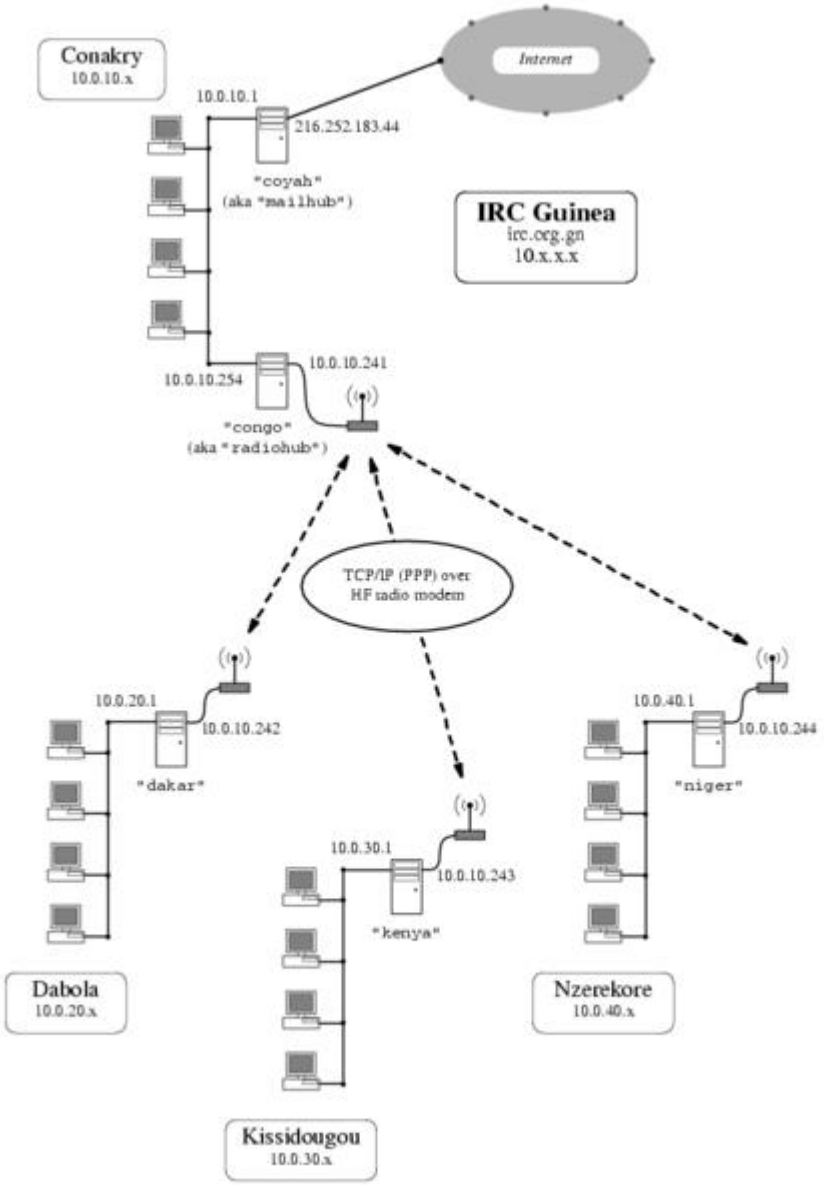


Figure 1. Radio Network Topology

For now, though, we don't have the luxury of dedicating our radios to full-time connections for data communications. In fact, voice communication continues to be the central purpose of the radio equipment. Our implementation and procedures must supplement this lifeline, not impair it. Because data sessions over the radio block voice calls at each end of the link, we have policies and configurations to hold connections to less than 15 minutes per session to keep the radios free for voice contact. In the field, radio operators have procedural protocols for making their periodic “dial-up” connections with Conakry for e-mail exchange at regular intervals throughout the day. Radio operators adjust the schedule and break sessions as necessary to accommodate urgent voice communications. For these reasons, all dial-ups are necessarily under the control of the radio operators, rather than set up with cron jobs or diald.

The equipment used in this project is made by Codan, an Australian manufacturer. Although there are other manufacturers, including Motorola, Kenwood and Yaesu, Codan seems to be the radio of choice for international aid organizations in this part of the world. Their big white Land Cruisers with official markings have substantial Codan whip antennas bolted conspicuously to every front bumper. The symbolic authoritative value of these thick black whips, such as when crossing the many military checkpoints, is certainly their most dominant feature.

The modem used in this project is their 9002 model. These modems are equipped with a basic Hayes-like AT command set, so they are easy to configure, operate and troubleshoot with any telecommunications application.

There are some significant differences between this modem and the family Sportster, however. For one thing, the Codan unit actually is built as DTE (data terminal equipment) rather than DCE (data communication equipment). To connect it to your serial port, you will need a DB-9 null modem cable, wired as diagrammed in David Lawler's Text-Terminal-HOWTO (www.tldp.org/HOWTO/Text-Terminal-HOWTO.html). Not all cables described as null modem are wired the same, so this detail is crucial. Also, the AT commands are not as extensive as, and vary slightly from, the standard command set. And, of course, this modem moves data more slowly than you can possibly imagine.

Configuring mgetty and PPP

The Conakry radio host congo, aliased as radiohub, is configured as a PPP server, ready to accept dial-in connections over the radio from offices in the field. As with a conventional telephone dial-in configuration, we use mgetty to watch the serial line, initialize the modem, wait for incoming calls, answer the “ring” and fire off the PPP dæmon.

The 9002 works pretty much as mgetty expects. We start by setting the modem to a known state with the factory defaults (AT&F) and then get defensively redundant. We set all command, local and remote echo off (E0 L0 R0); ignore carrier (X0); use hardware flow control (&K3); auto-answer off (S0=0 & A=0); and reset the station address (&I=*nnnn*, where *nnnn* is like a phone number, the unique identifier that other radios call to reach this particular station).

After mgetty answers the “ring” and gets a “connect” back from the the other end, then what? This is controlled by mgetty's login.conf file. It is common in dial-in systems to let mgetty look for incoming PPP packets and then start the PPP dæmon automagically, typically using CHAP in the link negotiation process for authentication. Such a configuration is handled by the line starting with the magic string **/AutoPPP/**.

In our experience, though, the high latency of the radio link makes an /AutoPPP/ configuration slow to kick in. What we do instead will be shocking: we dispense with conventional authentication entirely! In our configuration, the login name provided by the chat script of the incoming connection is used to start the PPP dæmon directly. When mgetty matches a login name with an entry in the first field of the login.conf file, such as Pklogin, it then runs the program specified in the fourth field, such as /usr/local/sbin/pppd.login.kenya. In essence, then, the login name used by the remote system serves to control access. Note that Pklogin is bogus as a system account, and you can be sure I haven't told you the string we are really using! (Note also that we have an implicit system of human authentication even before a connection is made, when operators agree by voice to lock on a specific channel before starting the link.)

When mgetty gets a login name listed in the login.conf file, it then passes control to the corresponding start-up script, such as pppd.login.kenya. This in turn starts the PPP dæmon, using an options file for the particular remote host, such as options.kenya.

To spare yourself bitter trials of sweat, tears and other emotional excretions, and pay attention to the lcp-restart and ipcp-restart options. These parameters give the time, in seconds, that pppd will wait to receive a reply to that particular phase of PPP negotiation before trying again. The default value of these parameters is three seconds, which is generally more than adequate when using regular telecommunications.

Over the radio, though, if these restart defaults are not extended, you'll only snag yourself a useless snarling hair ball. Here's what happens. As pppd starts up, each peer begins a negotiation process with the other to agree on all parameters for the connection. During these initial discussions, when one end of the link doesn't hear back from the other within its restart interval, pppd repeats the transmission. In the meantime, though, the remote end *has* received the original transmission and sends back its reply. The local end gets this response back to its first transmission, thinking it has a proper response to its second, and proceeds to the next step of negotiation. But then the local server gets what is now the unexpected response back from its *second* transmission, and the negotiations break down in unresolved chaos.

By extending the lcp-restart and ipcp-restart parameters, you can delay sending out repeat transmissions for a sufficient amount of time for the peers at each end to receive a response. We have configured a generous 16-second delay and have not had any more problems.

Turning our attention up-country, each remote host in the field is configured to dial up the radio server in Conakry. After all our testing, failures and ordeals of travel, it was a happy and amazing day when we finally got our first link across radio waves spreading invisibly through the Guinean atmosphere. From kenya we actually made SSH connections with congo simply to express our exuberance over “talk” sessions with the radio operators watching their terminal screen in Conakry—“Greetings from Kissi!”

With the configuration details finally worked out, we found that the PPP link, although slow, would come up reliably and remain stable at all times of the day, even over channels that otherwise sounded fuzzy and had considerable static. Of course, all radios and antennas should be tuned for their best performance. But once you are able to establish a link, it is reassuring to find that the radio modems are quite capable of maintaining it, even when conditions are less than optimal. Because somehow, conditions always have a way of being less than optimal.

qmail in the Bush

D. J. Bernstein, author of qmail, also has developed a number of special-purpose tools and applications perfectly suited to the Radio E-mail Project. Central among these, qmail includes a QMTP server, implementing Bernstein's own Quick Mail Transport Protocol. QMTP is a supplemental suite of programs designed for moving mail over slow connections.

As shown in the reference network in Figure 1, qmail runs on five hosts: the central mail server coyah, the radiohub server congo and each of the three field office hosts.

Once qmail considers the message delivered, we wait for the next PPP connection with the remote host kenya in Kissidougou. Then we can make use of the serialmail package to blast (relatively speaking) all the mail collected in the /var/qmail/qturn/kissidougou/.QMAIL.PPP/ mailbox across the link using QMTP.

Table 1. qmail Components

Each of the radio e-mail servers in the field run headlessly, controlled from a simple command-line interface via Telnet session from the operator's desktop PC. The basic interface consists of four commands, usually run in the following sequence:

```
ppp.start
mail.get
mail.send
ppp.stop
```

These commands are simple shell scripts that perform their respective tasks, each providing the operator with a modest amount of feedback about what is happening at the time. The functions could be further collected into a single command, such as mail.run, but we want to enable the operator to maintain some discretion over radio access, depending on the demands for voice communication.

We don't try to get and send mail simultaneously; we first do one, and then the other. This is another accommodation for the anemic, half-duplex bandwidth of the HF radio link. As far as network traffic goes, this link is like a one-lane back road—more than a little traffic creates a long skinny parking lot.

An Algorithm for Africa

As we may have mentioned, the HF radio link is a tad on the slow side. Nevertheless, we manage to move a decent amount of mail with it. On an average day, over 300 messages travel the air waves between Conakry and field offices, over two to three brief connections per office. And as is typical with all internet technologies, every taste stimulates an even greater appetite.

Given the limitations inherent in radio e-mail, we try to maintain a usage policy that is as open as possible. For example, staff members are free to use radio e-mail for personal correspondence with friends and family anywhere in the world, and there is no limit to the number of messages any user may send. Our only explicit policy restriction is the request that users not subscribe to mailing lists.

To prevent the radio links from getting choked-up for hours on huge attachments, such as large documents and graphic files, all qmail servers connected to radios (that is, the radiohub in Conakry and each of the field office servers) are run with a message size limit of 8,000 characters. This is sufficient for three to four pages of text. Whatever can be squeezed into the 8,000-byte limit by way of attachment and file compression is free to go.

The system has proven extremely reliable. Despite the intermittent power outages typical in Conakry, we do try to keep the mailhub server coyah running at all times by using a generator and battery backup. So far, these measures have kept this machine serving flawlessly since it was first installed, with a continuous uptime at this writing of over three months without reboot.

Yet this reliability would mean nothing if the system were not sustainable over the long term. Two months before we installed the first radio server in the field, we formed a Network Users/UNIX Group (IRC-NU/UG) among interested and capable IRC staff. This group meets regularly and enthusiastically to learn Linux/UNIX and to develop network administration skills. The group now has a

number of functional production systems on which to work and play, using mostly recycled hardware. The Linux servers installed for this project also host a typical range of other servers and services, including DHCP, DNS, NATD, Apache, FTP, Samba and PostgreSQL. The IRC-NU/UG provides a human network that will continue to sustain and grow the technical network in the years to come.

The successes of this project are readily duplicated anywhere in the world. Schools, government ministries and other organizations can build remote networking solutions easily over HF radio where access is otherwise not available and do it at a minimal cost. Once installed, these systems are almost trivial to administer and may be adapted quickly to alternative TCP/IP carriers. Maintenance of the e-mail system itself involves only the routine addition and deletion of user accounts and keeping the /etc/aliases files up-to-date.

We are now serving mail to over 50 desktops and 150 staff in four offices throughout Guinea. The entire wide area network is serviced behind a single public IP address, at a total ISP cost of $150 US per month. Best of all, the system has deployed standard network and internet technologies throughout the organization (and throughout Guinea) utilizing freely available technologies. Not only does this plant grassroots networking infrastructure where there is no Internet yet, it helps build the core competencies and capabilities essential for Africa's connected future.

Resources

Wayne Marshall (guinix@yahoo.com) is a UNIX programmer and technical consultant living in Guinea, who never intended to become an e-mail administrator. He and his wife Paula enjoy traveling and African skies.

Archive Index Issue Table of Contents

Advanced search

Introduction to Internationalization Programming

Olexiy Ye Tykhomyrov

Issue #103, November 2002

Olexiy describes the basic aspects of i18n and provides a sample i18n output program.

In the old days when only a few people used computers, they were, for the most part, English speakers. Today, computers are widely available, and differences in languages, traditions and cultures need to be reflected in the world of programming. This article introduces the GNU gettext system.

Internationalization and Localization

The idea of using the same program but changing its properties according to the cultural traditions of different peoples is called *internationalization*. However, because programmers like to make words shorter, instead of typing 20 characters they type only four: *i18n*. I18n means programming designed to handle many languages.

Once you've written an i18n program, you may want to add a new language. This is not an i18n problem. In general, you need a person who will translate messages from the program for a specific nation. This problem is called *localization*, or *i10n*. I10n refers to the implementation of a specific language for an internationalized software, or in other words, the creation of *localized objects* according to the specific region's rules.

Although each organization and company that designs and distributes software tries to implement this in its own way, in general, the i18n idea is simple. Software should be created with two parts in mind: common and nation-dependent. This second part is known as *localized objects*.

Hopefully, standards will make life more comfortable. The basic concept of *locale* was introduced by the ISO (International Standard Organization) with the C standard in 1990, which was expanded in 1995. POSIX also has rules for i18n,

so the term *POSIX locale* is used together with National Language Support (NLS). Formally, NLS is not a part of POSIX but has some functions that help when using the POSIX locale. X11 has its own i18n implementation, but the common way for programmers is to move the X11 i18n/i10n "up a level" into the POSIX/NLS locale. [Other software has its own i18n and i10n. See "Bridging the Digital Divide in South Africa" for one way to handle it].

Language, Coding and Charsets

What should we take into account when speaking of locale? Of course, the name of the language, but that is not enough. Everybody knows there are differences between American and British English, so we also have to know *where* the particular language is used, or in other words, the territory, taking into account individual traditions and cultural rules.

Every language has its own system of writing, and sometimes even several. Languages have alphabets, or character repertoires, but computers deal with digits. So, a character should be associated with a digit. This kind of association is called a coded character set (CCS). There are plenty of them, and each has its own name, such as ASCII, ISO-8859-1, KOI8-U. Instead of CCS, the term charset is often used. There is no special standard for the name of a charset, so ISO-8859-1, ISO8859-1 and iso8859-1 all refer to the same thing. There are some definitions from IANA, the organization that also is responsible for the registration of charsets (see Resources). As you probably know, the X11 system has its own system for charset naming, and their document "Logical Font Description Conversion" (described in Jim Flower's article, see Resources) provides a good name and alias charset creating system.

Charsets are important. Some countries have several different charsets for the same language! In the Ukraine, for instance, the same text may be displayed nicely in koi8-u but may be absolutely unreadable if a terminal uses the Ukrainian charsets iso8859-5, cp1251 or Unicode. In those cases, we would have to convert the text from one charset into another.

In order to take all of this into account, the POSIX locale defines some things that all together are called *locale categories*. They are shown in Table 1. Knowing them is important; C functions work differently with different locales! Categories are reflected in shell as environment variables with the same name. An example of using LC_ALL is shown in Listing 1.

Listing 1. Example of Using LC_ALL

The syntax to build a locale name looks like this:

```
language[_territory][.codeset][@modifier]
```

where *language* is represented by two lowercase letters, such as **en** for English and **fr** for French; *territory* is represented by two uppercase letters, such as **GB** for United Kingdom and **FR** for France, and in these two cases, **euro** would be the *modifier*. So, you can change your locale by setting the corresponding environment variables. See Listing 1, where we use the programs **date**, **cat** and our example program, **counter**. Note, we use only language and territory; we cannot change the charset for the terminal with this command. Now imagine that the program messages are written in one charset but are output in another. POSIX does not have functions to determine the current charset, but XPG has nl_langinfo(). In some distributions, the man page for this function may be missing (Debian does not have it, but SuSE and Red Hat do). In any case, you can obtain additional information from /usr/include/langinfo.h. To determine the current charset, use the following code:

```
#include <locale.h>
#include <langinfo.h>
 ...
 setlocale(LC_ALL,"");
 printf ("Current charset = %s\n",
            nl_langinfo(CODESET));
```

To convert from one coding into another for the correct output, you can use the conv() function. For more details, consult “Introduction to i18n” (see Resources).

In order to provide output information, a message catalog for that locale was created. This means that all software messages are kept separately from a program that may have (and must have) its own catalog. NLS provides a set of utilities for creating and supporting such catalogs, as well as functions for extracting information according to three keys: 1) program name, 2) current categories of locale and 3) pointer to a particular message to be output.

There are two general realisations of the NLS mechanism:

- **X/Open Portability Guide XPG3/XPG4/XPG5** with the functions catopen(), catgets() and catclose() and the gencat utility.
- **SUN XView** with functions gettext() and textdomain(). The GNU Project has its own fully compatible release called GNU gettext.

Usually programs, as well as system libraries, use one (or even two) NLS systems.

Although XPG5 is included in the UNIX specification version 2, and all versions of UNIX systems support it, with Linux, GNU gettext is the most popular solution.

The POSIX locale has the following components:

- Locale API, i.e., subroutines like setlocale(), isalpha(), etc.
- Shell environment variables to manage locale categories.
- The locale utility to get information about the current locale; see **man locale** for more details.
- Objects of localization. The default directory for their location is /usr/share/locale/.

Making a Program I18n-Compliant

If you are going to write a real i18n program, it would be wise to think that you know nothing about a specific language and take charsets into account. Ideographic languages have many more than 26 letters: Japanese has about 2,000, and Chinese has about 5,000. To deal with such characters, the POSIX locale has multibyte and Wide Class (wchar_t). The latter is done for Unicode. To convert one into another, functions like mblen(), mbstowcs(), wctomb(), mbtowc() and wcstombs() are used. However, using Unicode is beyond the scope of this article.

Producing real multilingual software is a complex task. Hopefully, the GNU gettext system that now conforms with SUN XView, will help you write i18n programs.

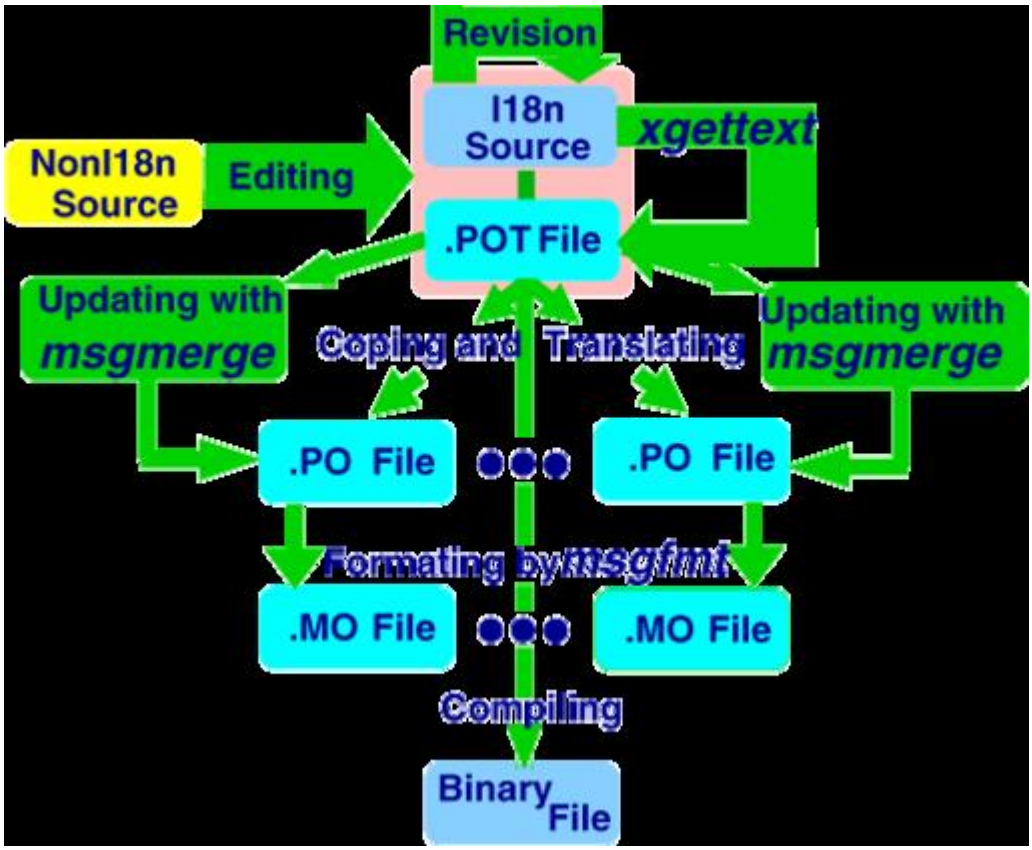


Figure 1. Producing an i18n Program

Figure 1 represents all necessary steps for producing an i18n program:

- To create an i18n version, you have to edit a non-i18n program. If you use a special editor mode you will create an additional file at the same time, called a POT file, where PO stands for portable object, and the letter T is for template.
- If you merely make a revision of an existing i18n program, or if a POT file does not exist, you have to use the xgettext program to produce it.
- Copy the template file into *ll*.po, where *ll* refers to a certain language.
- Translate messages into the language *ll*.
- Create a *ll*.mo file with the msgfmt program (mo stands for machine object). Sometimes you can see gmo files (g stands for GNU).
- Compile your source; put the binary program and *ll*.mo files into the right place. This and the previous steps are better accomplished with a Makefile.

Before looking briefly at all the steps of a simple program, please read these golden rules of internationalization:

1) Put the following lines into the non-executable part of your program, and mark messages in the source file as **_("*message*")** instead of **"*message*"** in the executable part of the program and **N_("*message*")** in the non-executable part. Pay attention to the output to guarantee passing the strings declared as constants through gettext, i.e., in the non-executable part:

```
#include <libintl.h>
#include <locale.h>
#define _(str) gettext (str)
#define gettext_noop(str) (str)
#define N_(str) gettext_noop (str)
```

2) Start your program by setting the locale:

```
setlocale (LC_ALL, "");
```

3) Indicate the message catalog name, and if necessary, its location:

```
bindtextdomain(PACKAGE,LOCALEDIR);
textdomain (PACKAGE);
```

PACKAGE and LOCALEDIR usually are provided either by config.h or by the Makefile file.

4) To check a symbol's properties and conversion, use calls like isalpha(), isupper(), ..., tolower() and toupper().

5) To compare strings, use the strcoll() and strxfrm() functions instead of strcmp().

6) To guarantee portability with old versions of locale, use a variable of type unsigned char for symbols, or compile your program with the -funsigned-char key.

Let's make a simple internationalized program in which these rules are ignored (Listing 2). The program outputs an invitation to type, reads a string from the terminal and counts the digits in it. The results of this counting are output in the terminal, then the program exits.

Listing 2. A Non-I18n Program

Because the program is small, we can change it easily according to the rules with your favorite editor; if the program is large, it is better to use special tools. Editors like (X)Emacs or vi with po mode can create a counter.pot file at the same time that you are changing the program source!

The changed file is shown in Listing 3. Lines 4-8 are added according to rule 1. Definitions from the locale.h file may not be necessary; they may be included within the libintl.h definitions. Writing gettext and gettext_noop many times is annoying, so we will use macros, as defined in lines 6-8. Using **gettext_noop** is an example of pre-initialized strings at the compile stage. A possible solution is shown in our program where using **gettext_noop** allows the strings to be recognized by gettext at the time of executing.

Listing 3. I18n Version of the Program Shown in Listing 2

Without line 15 (rule 2), the program will not understand your locale and will use the C locale. Note that sometimes it is necessary to set special categories of locale, such as LC_CTYPE and LC_MESSAGES. See **man setlocale** and Table 1 in this article for more information.

Table 1. Categories of Locle and Shell Variables

Lines 16 and 17 were inserted according to rule 3. Usually the parameters of these calls are provided in either a Makefile or a special file (like config.h) that holds configuration information, but in this program we put in the names directly. According to line 16, searching will be started in the current directory. If the line with the call is omitted, Linux will use the default location, /usr/share/locale.

The call textdomain() *must* be presented in any i18n program. It points the gettext system to the filename with i10n messages.

Lines 19, 25 and 26 where changed according to rule 1. Lines 19 and 25 are simple: instead of using strings directly, we call them through gettext to use a message catalog. Line 26 demonstrates the exception. We cannot transform strings defined in the non-executable part through gettext because there the values are initialized *before* running the program by the compiler. The problem is solved according to rule 1. We marked the strings with **N_** in line 12 to make them recognizable by xgettext; we used **_(mess)** instead of mess in line 26, as with normal strings. We do not need to do more, because of the function isdigit (see rule 4).

Now the program is internationalized. Compiling and running it, however, produces exactly the same result as the previous non-i18n one. Messages from the counter.pot file have to be translated into a specific language.

There is another way to create an initial .pot file. Once you have an i18n program, you can use xgettext. This scans the source files and creates corresponding strings for translation. In our case, we can invoke it like this:

```
xgettext -o counter.pot -k_ -kN_ counter.c
```

where **-o** is for output file name and **-k_ -kN_** is to extract strings that start with corresponding symbols. Consult **info xgettext** to get more details.

Basics of Localization

Typically, a programmer works on the source code, and a translator deals with the corresponding .po file, which may be created with the copy command from the .pot file or directly from the source with the xgettext program.

Glance over a .po file, and you will see it has a header and entries for translating. Here is an example of an entry:

```
# This is my own commentary
#: counter.c:25
#, c-format
msgid "You typed %d %s \n"
msgstr "Vous avez tapé %d %s\n"
```

It's simple. You translate phrases from msgid and put the results into the msgstr fields. If a line starts with **#:,** it is a reference to the source; if it starts with **#,** it shows an entry's attributes. You can add your own comments in lines starting with two symbols: **#** (the pound sign and then a white space).

After copying the template .pot file information (.po) or creating a new one with the xgettext command, you can start translating. Generally, this job can be done by another person using his or her favorite editor. (X)Emacs is not a bad choice for this job, but KBabel, part of KDE, is an even better one. If you are

going to participate in a team of translators, it is highly recommended that you use KBabel. Describing KBabel is beyond the scope of this article, but you can read more about it in “The KBabel Handbook” (see Resources).

Translation is a kind of art. Writing a correct phrase can be difficult, and you sometimes may doubt your ability with a particular language. So, you may want to leave some entries untranslated, or having translated them doubtfully, mark them as “fuzzy”. With KBabel or (X)Emacs, you easily can find such entries and edit them again later. Do not worry; only fully translated entries will be compiled later by msgfmt and become usable in programs. This simply means that an entry may be marked “translated”, “untranslated” or “fuzzy”, and as software changes quickly, there is also an “obsolete” attribute.

Languages are flexible. English messages are not always perfect either. In our case, the message “You typed 0 digit” is incorrect. GNU gettext can manage translating problems like word order, plural forms and ambiguities, but you have to use extra functions that hold more arguments than gettext().

Once you have translated the file, you should convert it into a .mo file that gettext will use if you run the program with the corresponding locale. Do not forget to put this file in the right place, in our case:

```
mv counter.mo fr/LCMESSAGES/
```

Now the counter can speak French! (See Listing 1.)

Managing a Message File

Programs evolve, and if their source code is changed, the corresponding .po files also have to be updated. Using only xgettext in this case is not an ideal solution. All translated messages will be lost, because it overwrites .po files. In this case, you should use the program msgmerge. This program merges two .po files, keeps translations already made (if the new strings match with the old), updates entries' attributes and adds new strings. Of course, these new strings will be untranslated entries. A typical call is simple:

```
msgmerge old.po new.po > up-to-date.po
```

Final Remarks

In this article, the input method is not described, although it is also important. Generally, non-X11 software doesn't need to worry about i18n input methods, because it is the responsibility of the console and X terminal emulators.

For input in X11, three methods exist: Xsi, Ximp and XIM. The first two are old-fashioned; the last one is the de facto standard. Their descriptions are beyond

the scope of this article; however, the source code for the rxvt program provides an excellent example.

Modern tools provide their own special subroutines for input of internationalized strings, using gettext for output messages. To make program code 8-bit transparent for internal proposals, Unicode is used. Qt, for instance, works in such a way, providing additional functions for input and output of i18n strings correctly (see Resources).

You also may want to look at the source code of mutt, which is a good i18n program (www.mutt.org). This program supports aliases for charsets.

Using Unicode in your programs is described by Tomohiro Kubota (see Resources). Happy i18ning!

Resources

Olexiy Tykhomyrov (tiger@ff.dsu.dp.ua) has been using Linux since 1994, after visiting ICTP (International Center for Theoretical Physics) Real-Time Courses. He works for the Department of Experimental Physics at the Dnepropetrovsk National University and teaches physics and communications.

Archive Index Issue Table of Contents

Advanced search

Indian Language Solutions for GNU/Linux

Frederick Noronha

Issue #103, November 2002

Some of the world's most widely spoken languages are the hardest to support on a computer. Here's how support is coming together for Hindi, Malayalam and other languages of the Indian subcontinent.

South Asia, home to nearly one-sixth of humanity, is struggling to attain regional language solutions that would make computing accessible to everyone. Even if most are poor and have low purchasing ability, this could open the floodgate to greater computing power and much-needed efficiency in a critical area of the globe. However, some call Indic and other South Asian scripts the final challenge for full-i18n support.

Some Indian regional languages are larger than those spoken by whole countries elsewhere. Hindi, with 366 million speakers, is second only to Mandarin Chinese. Telugu has 69 million; Marathi, 68 million; and Tamil, 66 million. Sixteen of the top 70 global languages are Indian languages with more than 10 million speakers. Other languages spoken in India are also spoken elsewhere. Bengali has 207 million speakers in India and Bangladesh, and Urdu has 60 million in Pakistan and India.

Simputer Offers Text-to-Speech

The Simputer is a simple and relatively inexpensive Linux computer for people in Indian villages. The creation of the Simputer is being organized with a hardware license, the Simputer General Public License, modeled on the GPL. Although the license provides for free publication of specifications, it does require a one-time royalty payment before licensees sell Simputers.

The Simputer features a 200MHz StrongARM processor, 32MB of DRAM, 24MB of Flash storage, a monochrome display, speaker and microphone.

dhvani is a text-to-speech system for Indian languages developed by the Simputer Trust developers and others. It is promising to have a better phonetic engine, Java port and language-independent framework soon. (See sourceforge.net/projects/dhvani.) Meanwhile, IMLI is a browser created by the Simputer Trust for the IML markup language. It is designed for easy creation of Indian language content and is integrated with the text-to-speech engine.

I18n Frameworks Already in Place

In Kerala, a southern state with an impressive 90% literacy rate whose language Malayalam is spoken by 35 million people, senior local government official Ajay Kumar (kumarajay1111@yahoo.com) is leading an initiative to make GNU/Linux Malayalam-friendly: "We propose to develop a renderer for our language. Specifically, we are looking for a renderer for Pango (the generic engine used with the GTK toolkit)."

He adds, that in nine months time, "we want to create an atmosphere where language computing in Malayalam improves." He also says, "We are confident that once we deliver the basic framework, others will start localizing more applications in Malayalam."

At the toolkit level, GTK and Qt are the most used. GTK already has a good framework through the Pango Project and has basic support for Indian languages. Qt also now has Unicode support for all languages, but rendering is not yet ready.

International efforts also are helping India. Yudit, the free Unicode text editor, now offers support for three South Indian languages: Malayalam, Kannada and Telugu. Delhi-based GNU/Linux veteran Raj Mathur commented, "The current version of Yudit has complete support for Malayalam and other Indic

languages. It can also use OpenType layout tables of Malayalam fonts. I think Yudit is the first application that can use OpenType tables for Malayalam."

K Ratheesh was a student of the Indian Institute of Technology-Madras (at the South Indian town of Chennai) when he worked on enabling the GNU/Linux console for local languages a couple years ago. He said:

> As the [then] current PSF format didn't support variable width fonts, I have made a patch in the console driver so that it will load a user-defined multiglyph mapping table so that multiple glyphs can be displayed for a single character code. All editing operations also will be taken care of.

In Indian languages, there are various consonant/vowel modifiers that result in complex character clusters. "So I have extended the patch to load user-defined, context-sensitive parse rules for glyphs and character codes as well. Again, all editing operations will behave according to the parse rule specifications", Ratheesh commented.

Ratheesh also said, "Even though the patch has been developed keeping Indian languages in mind, I feel it will be applicable to many other languages (such as Chinese) that require wider fonts on console or user-defined parsing at I/O level."

The package, containing the patch, some documentation, utilities and sample files then weighed in at around 100KB.

Which Languages First?

Many issues need to be tackled in the search for solutions. For instance, which languages need be tackled first? Joseph Koshy (JKoshy@FreeBSD.ORG), HP's Bangalore-based technical consultant, argues that the North Indian Hindi family promises the greatest reach population-wise. However, he feels the southern languages—Kannada, Telugu, Tamil and Malayalam—offer the greatest promise of real-world deployability. They enjoy a better support infrastructure needed to deploy an effective IT solution, which appears to be better in South India.

Outside of his work life at HP, Koshy is a volunteer developer of the FreeBSD operating system and one of the founders of the Indic-Computing Project on SourceForge. Koshy says:

> What I am interested in is helping make standards-based, interoperable computing for Indian languages a reality. This dream is bigger than any one operating system or any one computing platform. I want to see

> pagers, telephones, PDAs and other devices that have not been invented yet interacting with our people in our native languages.

But others have different views. CV Radhakrishnan (cvr@river-valley.org), a TeX programmer who has contributed packages to the Comprehensive TeX Archive Network and runs River Valley Technologies in South India, says:

> I think most of the South Indian languages could pose problems [because of] their nonlinear nature. For example, to create conjunct glyphs one has to go back and forth, while North Indian languages do not have this problem. Malayalam has peculiar characters called half consonants ["chillu"]. There is no equivalent for this in other languages. This raises severe computing/ programming challenges.

Koshy notes that the official Census of India lists 114 "major" languages in the subcontinent, and that linguists, who discriminate more finely than the Census officials, peg the number of living languages in India at 850 plus.

According to Koshy, out of the 18 more important "scheduled" national languages, all except the Devanagari-based ones (which use the same script as Hindi) have serious issues when it comes to representing and processing them on a computer.

Other OSes Lag

Proprietary OSes have trouble with Indian language support too, says Edward Cherlin (edward@webforhumans.com) who creates multilingual web sites: "Progress is slow at Microsoft and Apple. Linux should pass them by the end of the year, or early in 2003."

On GNU/Linux, Cherlin points out, you can volunteer to Indicize any application. In the future, when font management and rendering are standardized, all applications will run in Indian languages for input and output without further ado, and anyone will be able to create a localization file to customize the user interface. According to Cherlin, volunteers also are needed to translate documentation.

Additionally, Cherlin says, "Apple and Microsoft are not willing simply to support typing, display and printing. They will not release language and writing system support until they have complete locales built, preferably including a dictionary and spell checker." Linux is under no such constraints.

He points out that the Free Standards Group together with Li18nux.org are proposing to rationalize and simplify i18n support under X, including a common

rendering engine, shared font paths and other standards that will simplify greatly the business of supporting all writing systems and all languages. Cherlin feels that Yudit and Emacs both support several Indic scripts and could be extended with only moderate effort on the part of a few experts.

Microsoft's Windows XP has Indian language support based on the current Unicode version (3.x) and hence suffers from all the problems of Unicode-based solutions: inability to represent all the characters of some Indian languages and awkwardness in text processing.

“When Microsoft came up with the South Asian edition of MS-Word, the fonts had a lot of problems. Mostly, words were rendered as separate letters with space in between and not combined together as is the case with most Indian languages”, says PicoPeta language technology specialist Kalika Bali. PicoPeta is one of the firms working to create the Simputer.

However, Dr UB Pavanaja (pavanaja@vishvakannada.com), a former scientist now widely noticed for his determined work to push computing in the influential South Indian language of Kannada (www.vishvakannada.com), sees the progress as being “quite remarkable, compared with the scene about two years ago”. Pavanaja says, “Current pricing and product activation of [Windows] XP may become a boon for GNU/Linux.”

Mandrake Linux includes Bengali, Gujarati, Gurmukhi, Hindi Devanagari and Tamil out of the box. That leaves Oriya, Malayalam, Telugu and Kannada yet to be done, along with the Indic-derived Lao, Sinhala, Myanmar and Khmer. Tibetan and Thai are moderately well supported, Cherlin contends.

“Recently, localization efforts are picking up”, agrees Nagarajuna G, scientist/ free software advocate in Mumbai. He also notes:

> If the government or other funding agencies can spare any amount to bodies like the Free Software Foundation of India and others who are active in the localization initiative, the developers could work with obsession and make this happen very fast. FSFIndia is presently working with the Kerala government to produce Malayalam support for the GNOME desktop.

Projects Finding an Indian Tongue for the Penguin

Font Challenges Remain

For desktop-class machines, current font technology (TTF, OpenType, Type 1, etc.) is capable of handling Indic scripts. Availability of quality fonts is another matter, but as Koshy puts it, this is not really a showstopper. Display technology

for embedded devices, such as pagers and other small devices, for Indian languages is not well developed.

Languages like Urdu and Sindhi have right-to-left scripts that look similar to Arabic but are, in fact, different, says Prakash Advani who some years back launched the FreeOS.com initiative. Urdu is the main language of Pakistan but is also used in India.

Satish Babu (sb@inapp.com), a free software enthusiast and vice president of InApp, an Indo-US software company dealing with free and open-source solutions, points to other problems, such as collation (sorting) order confusion (oftentimes, there is no unique "natural" collation order, and one has to be adopted through standardization).

There's also the non-availability of dictionaries and thesauri in Indian languages and issues arising out of multiple correct spellings for words; encoding standardization (Unicode) that will, inter alia, facilitate transliteration between Indian languages programme support (database, spreadsheet) for sorting/ searching two-byte strings; lack of support of some languages (e.g., Tulu, Konkani, Haryanvi and Bhojpuri), which are the mother tongues for some sections of our population.

Ravikant (ravikant@sarai.net), who taught history at Delhi University before moving to the Language and New Media Project of sarai.net (www.sarai.net), says, "The long-term solution is of course Unicode." For short-term measures, he suggests working toward developing the existing packages "in a manner that people can use them with freedom from OSes and fonts". ITRANS and WRITE32, written by Indians settled abroad, are transliteration packages that already do so. The LaTeX-Devnag package is being used and promoted by Mahatma Gandhi International University, Delhi.

Says Prakash Advani (prakash@netcore.co.in) of the FreeOS.com initiative: "There is definitely a market for Indian language computing that exists today, but there is a huge untapped market: 95% of the population doesn't read or write English. If we can provide a low-cost Indian language computer, it will be a killer."

Advani also says, "The biggest challenge is not technical but a lack of standards. Till Unicode happened, there was a complete lack of standards. Everyone was following their own standards of input, storage and output of data."

There is a lack of free Indian language fonts. "There are over 5,000 commercial Indian language fonts, but there are probably ten Free (GPL/royalty free) Indian

language fonts. This is a serious issue, and more efforts should be made to release free fonts", Advani adds.

Besides, others point out, fonts are another mess altogether. Most of the current implementations rely on glyph locations to display and store information. For instance, to represent the letter "a", what is stored is the position of "a" in some particular font used by that package. This is different from normal English where the ASCII standard specifies that to represent "a" the number 65 must be used. No such standard exists for Indian languages, and thus one document written in one application cannot be opened in another. This is also the reason why authors of Indian web pages must specify particular fonts.

Vendors use this situation to lock in their customers to a particular product. This also limits the exchange of e-mail to only situations where both parties have the same web interface or program for using an Indian language e-mail.

TUGIndia, which Raj Mathur represents, has procured a Malayalam font (Keli) from font designer Hashim. The font is currently available at www.linuxense.com/oss/keli.volt.zip. This is the OpenType source for the Malayalam font. You can edit and improve the font tables with Microsoft VOLT. The font tables are released under the GNU GPL, and the glyph shapes can be changed provided that the fonts are redistributed under a different name. Raj works as an engineer at Linuxense Information Systems and leads the Indian TeX users' group's localization project.

G Karunakar, another young developer with a keen interest in this field, says, "There are very few people in India who understand font technology completely, so most available fonts are buggy. Due to a lack of font standard, our fonts are not tagged as Indian language fonts."

Right now, a general consensus seems to be building on OpenType fonts as the suitable technology for Indian language fonts. There is already a free Devanagari font ("Raghu" by Dr RK Joshi, NCST and used in Indix), a Kannada OpenType from KGP, also for Malayalam, Telugu and Bengali.

The Indian TeX Users Group currently has a project to fund font designers in all the Indian languages who are ready to write fonts and release them under GPL. They've thus secured "Keli" a Malayalam font family in various weights and shapes written by Hashim and released under GPL. "We do hope to get more fonts in other languages to fill in the gaps. We hope to use the savings generated with TUG2002 (which was held in India in September 2002) exclusively for this purpose", says Radhakrishnan in Thiruvananthapuram.

“A lot of the know-how exists in rare books, which are difficult to get. A lot of research work done by scholars, linguists, typographers, etc., is going untapped, because we don't know of it, or the people who know it,” adds Karunakar.

The Future

Shrinath (shrinath@konark.ncst.ernet.in), a senior staff scientist at Mumbai's NCST, which has done some interesting work on this subject, says:

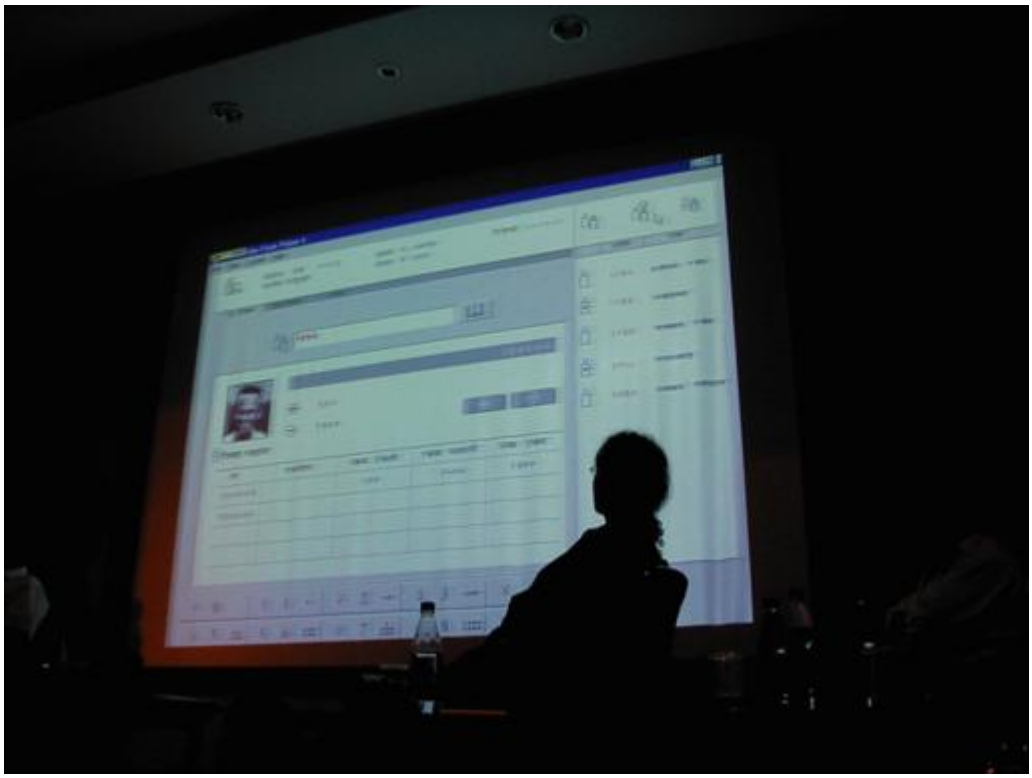
> We want Indian language programming to be as simple as programming in English is today. Almost every company has to re-invent the wheel or buy costly solutions from others. In English, the OS supports it. It's a chicken-and-egg problem. If there are applications in Indic, OS vendors will build the fundamental capabilities into the OS, and if the capabilities are built in, there will be more applications.

“English has been the de facto language for software development as well as usage. So there is a long way to go. China is working quickly on that end, and so can we”, argues Girish S, an electronics engineer from the central Indian region of Madhya Pradesh, who set up ApnaJabalpur.com.

There are other needs too, such as dictionaries and spell checkers. Word breaking doesn't operate the same way in Indic scripts as in the Latin alphabet and neither does fine typography, which you don't find in consumer or office applications in any language. Finally, the main challenge is the sheer numbers. India is believed to have 1,652 mother tongues, of which, 33 are spoken by people numbering over a 100,000.

At the time of this writing, a number of key proponents of Indianization are planning to meet in the southern city of Bangalore at the end of September to discuss local language development tools, applications and content. The meeting is planned for 20-25 core participants.

“We also hope that this broad coalition would play a facilitatory role in helping local language groups interact more effectively with international standards processes and forums, such as the Unicode Consortium and W3C”, says Tapan Parikh, one of the organisers.



Tapan Parikh is one of the prganisers of an Indianization conference scheduled for Bangalore in September 2002.

email: wccs@goatelecom.com

Frederick Noronha (fred@bytesforall.org) is a freelance journalist working out of Goa, India. He has been closely reporting on the GNU/Linux scene in the world's second-most populous country and was awarded a print-media fellowship from sarai.net last year to study open-source and free software issues in the region.

Archive Index Issue Table of Contents

Advanced search

Playing with ptrace, Part I

Pradeep Padala

Issue #103, November 2002

Using ptrace allows you to set up system call interception and modification at the user level.

Have you ever wondered how system calls can be intercepted? Have you ever tried fooling the kernel by changing system call arguments? Have you ever wondered how debuggers stop a running process and let you take control of the process?

If you are thinking of using complex kernel programming to accomplish tasks, think again. Linux provides an elegant mechanism to achieve all of these things: the ptrace (Process Trace) system call. **ptrace** provides a mechanism by which a parent process may observe and control the execution of another process. It can examine and change its core image and registers and is used primarily to implement breakpoint debugging and system call tracing.

In this article, we learn how to intercept a system call and change its arguments. In Part II of the article we will study advanced techniques—setting breakpoints and injecting code into a running program. We will peek into the child process' registers and data segment and modify the contents. We will also describe a way to inject code so the process can be stopped and execute arbitrary instructions.

Basics

Operating systems offer services through a standard mechanism called system calls. They provide a standard API for accessing the underlying hardware and low-level services, such as the filesystems. When a process wants to invoke a system call, it puts the arguments to system calls in registers and calls soft interrupt 0x80. This soft interrupt is like a gate to the kernel mode, and the kernel will execute the system call after examining the arguments.

On the i386 architecture (all the code in this article is i386-specific), the system call number is put in the register %eax. The arguments to this system call are put into registers %ebx, %ecx, %edx, %esi and %edi, in that order. For example, the call:

```
write(2, "Hello", 5)
```

roughly would translate into

```
movl   $4, %eax
movl   $2, %ebx
movl   $hello,%ecx
movl   $5, %edx
int    $0x80
```

where $hello points to a literal string “Hello”.

So where does ptrace come into picture? Before executing the system call, the kernel checks whether the process is being traced. If it is, the kernel stops the process and gives control to the tracking process so it can examine and modify the traced process' registers.

Let's clarify this explanation with an example of how the process works:

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <linux/user.h>   /* For constants
                                ORIG_EAX etc */
int main()
{   pid_t child;
    long orig_eax;
    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execl("/bin/ls", "ls", NULL);
    }
    else {
        wait(NULL);
        orig_eax = ptrace(PTRACE_PEEKUSER,
                          child, 4 * ORIG_EAX,
                          NULL);
        printf("The child made a "
               "system call %ld\n", orig_eax);
        ptrace(PTRACE_CONT, child, NULL, NULL);
    }
    return 0;
}
```

When run, this program prints:

```
The child made a system call 11
```

along with the output of ls. System call number 11 is execve, and it's the first system call executed by the child. For reference, system call numbers can be found in /usr/include/asm/unistd.h.

As you can see in the example, a process forks a child and the child executes the process we want to trace. Before running **exec**, the child calls ptrace with the first argument, equal to PTRACE_TRACEME. This tells the kernel that the process is being traced, and when the child executes the execve system call, it hands over control to its parent. The parent waits for notification from the kernel with a wait() call. Then the parent can check the arguments of the system call or do other things, such as looking into the registers.

When the system call occurs, the kernel saves the original contents of the eax register, which contains the system call number. We can read this value from child's USER segment by calling ptrace with the first argument PTRACE_PEEKUSER, shown as above.

After we are done examining the system call, the child can continue with a call to ptrace with the first argument PTRACE_CONT, which lets the system call continue.

ptrace Parameters

ptrace is called with four arguments:

```
long ptrace(enum __ptrace_request request,
            pid_t pid,
            void *addr,
            void *data);
```

The first argument determines the behaviour of ptrace and how other arguments are used. The value of request should be one of PTRACE_TRACEME, PTRACE_PEEKTEXT, PTRACE_PEEKDATA, PTRACE_PEEKUSER, PTRACE_POKETEXT, PTRACE_POKEDATA, PTRACE_POKEUSER, PTRACE_GETREGS, PTRACE_GETFPREGS, PTRACE_SETREGS, PTRACE_SETFPREGS, PTRACE_CONT, PTRACE_SYSCALL, PTRACE_SINGLESTEP, PTRACE_DETACH. The significance of each of these requests will be explained in the rest of the article.

Reading System Call Parameters

By calling ptrace with PTRACE_PEEKUSER as the first argument, we can examine the contents of the USER area where register contents and other information is stored. The kernel stores the contents of registers in this area for the parent process to examine through ptrace.

Let's show this with an example:

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <linux/user.h>
#include <sys/syscall.h>   /* For SYS_write etc */
int main()
```

```
{   pid_t child;
    long orig_eax, eax;
    long params[3];
    int status;
    int insyscall = 0;
    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execl("/bin/ls", "ls", NULL);
    }
    else {
       while(1) {
          wait(&status);
          if(WIFEXITED(status))
              break;
          orig_eax = ptrace(PTRACE_PEEKUSER,
                     child, 4 * ORIG_EAX, NULL);
          if(orig_eax == SYS_write) {
             if(insyscall == 0) {
                /* Syscall entry */
                insyscall = 1;
                params[0] = ptrace(PTRACE_PEEKUSER,
                                   child, 4 * EBX,
                                   NULL);
                params[1] = ptrace(PTRACE_PEEKUSER,
                                   child, 4 * ECX,
                                   NULL);
                params[2] = ptrace(PTRACE_PEEKUSER,
                                   child, 4 * EDX,
                                   NULL);
                printf("Write called with "
                       "%ld, %ld, %ld\n",
                       params[0], params[1],
                       params[2]);
                }
          else { /* Syscall exit */
                eax = ptrace(PTRACE_PEEKUSER,
                             child, 4 * EAX, NULL);
                    printf("Write returned "
                           "with %ld\n", eax);
                    insyscall = 0;
                }
            }
            ptrace(PTRACE_SYSCALL,
                   child, NULL, NULL);
        }
    }
    return 0;
}
```

This program should print an output similar to the following:

```
ppadala@linux:~/ptrace > ls
a.out        dummy.s      ptrace.txt
libgpm.html  registers.c  syscallparams.c
dummy        ptrace.html  simple.c
ppadala@linux:~/ptrace > ./a.out
Write called with 1, 1075154944, 48
a.out        dummy.s      ptrace.txt
Write returned with 48
Write called with 1, 1075154944, 59
libgpm.html  registers.c  syscallparams.c
Write returned with 59
Write called with 1, 1075154944, 30
dummy        ptrace.html  simple.c
Write returned with 30
```

Here we are tracing the write system calls, and **ls** makes three write system calls. The call to ptrace, with a first argument of PTRACE_SYSCALL, makes the kernel stop the child process whenever a system call entry or exit is made. It's

equivalent to doing a PTRACE_CONT and stopping at the next system call entry/exit.

In the previous example, we used PTRACE_PEEKUSER to look into the arguments of the write system call. When a system call returns, the return value is placed in %eax, and it can be read as shown in that example.

The status variable in the wait call is used to check whether the child has exited. This is the typical way to check whether the child has been stopped by ptrace or was able to exit. For more details on macros like WIFEXITED, see the wait(2) man page.

Reading Register Values

If you want to read register values at the time of a syscall entry or exit, the procedure shown above can be cumbersome. Calling ptrace with a first argument of PTRACE_GETREGS will place all the registers in a single call.

The code to fetch register values looks like this:

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <linux/user.h>
#include <sys/syscall.h>
int main()
{   pid_t child;
    long orig_eax, eax;
    long params[3];
    int status;
    int insyscall = 0;
    struct user_regs_struct regs;
    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execl("/bin/ls", "ls", NULL);
    }
    else {
       while(1) {
          wait(&status);
          if(WIFEXITED(status))
              break;
          orig_eax = ptrace(PTRACE_PEEKUSER,
                            child, 4 * ORIG_EAX,
                            NULL);
          if(orig_eax == SYS_write) {
              if(insyscall == 0) {
                 /* Syscall entry */
                 insyscall = 1;
                 ptrace(PTRACE_GETREGS, child,
                        NULL, &regs);
                 printf("Write called with "
                        "%ld, %ld, %ld\n",
                        regs.ebx, regs.ecx,
                        regs.edx);
              }
              else { /* Syscall exit */
                  eax = ptrace(PTRACE_PEEKUSER,
                               child, 4 * EAX,
                               NULL);
                  printf("Write returned "
                         "with %ld\n", eax);
```

```
                insysscall = 0;
            }
        }
        ptrace(PTRACE_SYSCALL, child,
               NULL, NULL);
    }
  }
  return 0;
}
```

This code is similar to the previous example except for the call to ptrace with PTRACE_GETREGS. Here we have made use of the user_regs_struct defined in <linux/user.h> to read the register values.

Doing Funny Things

Now it's time for some fun. In the following example, we will reverse the string passed to the write system call:

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <linux/user.h>
#include <sys/syscall.h>
const int long_size = sizeof(long);
void reverse(char *str)
{   int i, j;
    char temp;
    for(i = 0, j = strlen(str) - 2;
        i <= j; ++i, --j) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
}
void getdata(pid_t child, long addr,
             char *str, int len)
{   char *laddr;
    int i, j;
    union u {
            long val;
            char chars[long_size];
    }data;
    i = 0;
    j = len / long_size;
    laddr = str;
    while(i < j) {
        data.val = ptrace(PTRACE_PEEKDATA,
                          child, addr + i * 4,
                          NULL);
        memcpy(laddr, data.chars, long_size);
        ++i;
        laddr += long_size;
    }
    j = len % long_size;
    if(j != 0) {
        data.val = ptrace(PTRACE_PEEKDATA,
                          child, addr + i * 4,
                          NULL);
        memcpy(laddr, data.chars, j);
    }
    str[len] = '\0';
}
void putdata(pid_t child, long addr,
             char *str, int len)
{   char *laddr;
    int i, j;
    union u {
            long val;
            char chars[long_size];
```

```
    }data;
    i = 0;
    j = len / long_size;
    laddr = str;
    while(i < j) {
        memcpy(data.chars, laddr, long_size);
        ptrace(PTRACE_POKEDATA, child,
               addr + i * 4, data.val);
        ++i;
        laddr += long_size;
    }
    j = len % long_size;
    if(j != 0) {
        memcpy(data.chars, laddr, j);
        ptrace(PTRACE_POKEDATA, child,
               addr + i * 4, data.val);
    }
}
int main()
{
   pid_t child;
   child = fork();
   if(child == 0) {
      ptrace(PTRACE_TRACEME, 0, NULL, NULL);
      execl("/bin/ls", "ls", NULL);
   }
   else {
      long orig_eax;
      long params[3];
      int status;
      char *str, *laddr;
      int toggle = 0;
      while(1) {
         wait(&status);
         if(WIFEXITED(status))
             break;
         orig_eax = ptrace(PTRACE_PEEKUSER,
                           child, 4 * ORIG_EAX,
                           NULL);
         if(orig_eax == SYS_write) {
            if(toggle == 0) {
               toggle = 1;
               params[0] = ptrace(PTRACE_PEEKUSER,
                                  child, 4 * EBX,
                                  NULL);
               params[1] = ptrace(PTRACE_PEEKUSER,
                                  child, 4 * ECX,
                                  NULL);
               params[2] = ptrace(PTRACE_PEEKUSER,
                                  child, 4 * EDX,
                                  NULL);
               str = (char *)calloc((params[2]+1)
                                    * sizeof(char));
               getdata(child, params[1], str,
                       params[2]);
               reverse(str);
               putdata(child, params[1], str,
                       params[2]);
            }
            else {
               toggle = 0;
            }
         }
      ptrace(PTRACE_SYSCALL, child, NULL, NULL);
      }
   }
   return 0;
}
```

The output looks like this:

```
ppadala@linux:~/ptrace > ls
a.out        dummy.s      ptrace.txt
libgpm.html  registers.c  syscallparams.c
dummy        ptrace.html  simple.c
```

```
ppadala@linux:~/ptrace > ./a.out
txt.ecartp      s.ymmud      tuo.a
c.sretsiger     lmth.mpgbil  c.llacys_egnahc
c.elpmis        lmth.ecartp  ymmud
```

This example makes use of all the concepts previously discussed, plus a few more. In it, we use calls to ptrace with PTRACE_POKEDATA to change the data values. It works exactly the same way as PTRACE_PEEKDATA, except it both reads and writes the data thatt the child passes in arguments to the system call whereas PEEKDATA only reads the data.

Single-Stepping

ptrace provides features to single-step through the child's code. The call to ptrace(PTRACE_SINGLESTEP,..) tells the kernel to stop the child at each instruction and let the parent take control. The following example shows a way of reading the instruction being executed when a system call is executed. I have created a small dummy executable for you to understand what is happening instead of bothering with the calls made by libc.

Here's the listing for dummy1.s. It's written in assembly language and compiled as gcc -o dummy1 dummy1.s:

```
.data
hello:
    .string "hello world\n"
.globl  main
main:
    movl    $4, %eax
    movl    $2, %ebx
    movl    $hello, %ecx
    movl    $12, %edx
    int     $0x80
    movl    $1, %eax
    xorl    %ebx, %ebx
    int     $0x80
    ret
```

The example program that single-steps through the above code is:

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <linux/user.h>
#include <sys/syscall.h>
int main()
{   pid_t child;
    const int long_size = sizeof(long);
    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execl("./dummy1", "dummy1", NULL);
    }
    else {
        int status;
        union u {
            long val;
            char chars[long_size];
        }data;
        struct user_regs_struct regs;
        int start = 0;
```

```
            long ins;
            while(1) {
                wait(&status);
                if(WIFEXITED(status))
                    break;
                ptrace(PTRACE_GETREGS,
                       child, NULL, &regs);
                if(start == 1) {
                    ins = ptrace(PTRACE_PEEKTEXT,
                                 child, regs.eip,
                                 NULL);
                    printf("EIP: %lx Instruction "
                           "executed: %lx\n",
                           regs.eip, ins);
                }
                if(regs.orig_eax == SYS_write) {
                    start = 1;
                    ptrace(PTRACE_SINGLESTEP, child,
                           NULL, NULL);
                }
                else
                    ptrace(PTRACE_SYSCALL, child,
                           NULL, NULL);
            }
        }
        return 0;
    }
```

This program prints:

```
hello world
EIP: 8049478 Instruction executed: 80cddb31
EIP: 804947c Instruction executed: c3
```

You might have to look at Intel's manuals to make sense out of those instruction bytes. Using single stepping for more complex processes, such as setting breakpoints, requires careful design and more complex code.

In Part II, we will see how breakpoints can be inserted and code can be injected into a running program.

All of the example code from this article and from Part II (which will be printed in next month's issue) is available as a tar archive on the *Linux Journal* FTP site [ftp.linuxjournal.com/pub/lj/listings/issue103/6011.tgz].

email: ppadala@cise.ufl.edu

Pradeep Padala is currently working on his Master's degree at the University of Florida. His research interests include Grid and distributed systems. He can be reached via e-mail at p_padala@yahoo.com or through his web site (www.cise.ufl.edu/~ppadala).

Archive Index Issue Table of Contents

Advanced search

QUORUM: Prepaid Internet at the University of Zululand

Soren Aalto

Issue #103, November 2002

Using Squid's URL redirection to implement WWW usage quotas.

The University of Zululand is a “historically disadvantaged” university campus with approximately 6,000 students located in the northeastern coastal region of South Africa. Our disadvantaged legacy continues in post-apartheid South Africa, and our students are mostly black and are from financially disadvantaged backgrounds. Consequently, our operating budgets are severely constrained, and the daily challenge of doing more with less prevails.

In this environment, Linux is the obvious choice for our internet services platform, and we use it for practically everything: e-mail and WWW servers, DNS, DHCP, HTTP proxies, firewalling and dial-up access. Our management loves the “free-as-in-beer” aspect of Linux. However, our main reason for using Linux is more compelling—using Linux is the most fun way to deliver our internet services.

Cost of Internet Access

Early in 2000, our biggest financial challenge was to provide internet access to all our staff and students. The cost of internet bandwidth in South Africa is much higher than in the US. Our 128Kbps access circuit was costing approximately $5,000 US per month. This circuit already was saturated during the day by our 400 staff users, and we still needed to provide internet access to the 350 workstations in our heavily used student labs.

We needed more bandwidth, but our budget wouldn't cover doubling or tripling our access bandwidth. Without monitoring or placing controls on internet usage, management was uneasy about committing to large increases in spending on internet access. They were skeptical of paying large monthly bills when they didn't know who was using the Internet or for what purpose.

Quotas and Prepayment

Our problem was really one of how to provide an acceptable quality of service (QoS) for WWW browsing within our budgetary constraints. With unregulated internet access, congestion of the access circuit is the only factor limiting demand, which usually leads to the poorest QoS that the core group of die-hard downloaders can bear.

We needed to provide some type of “cost” for internet usage in order to regulate demand. This system would educate our users about the real costs of providing internet access at the university. For students, this cost should be prepaid. The level of bad tuition debt at disadvantaged universities in South Africa is a significant problem, and we didn't want to add bills for internet access to the debt collection problem. We needed a *quota* system for internet usage.

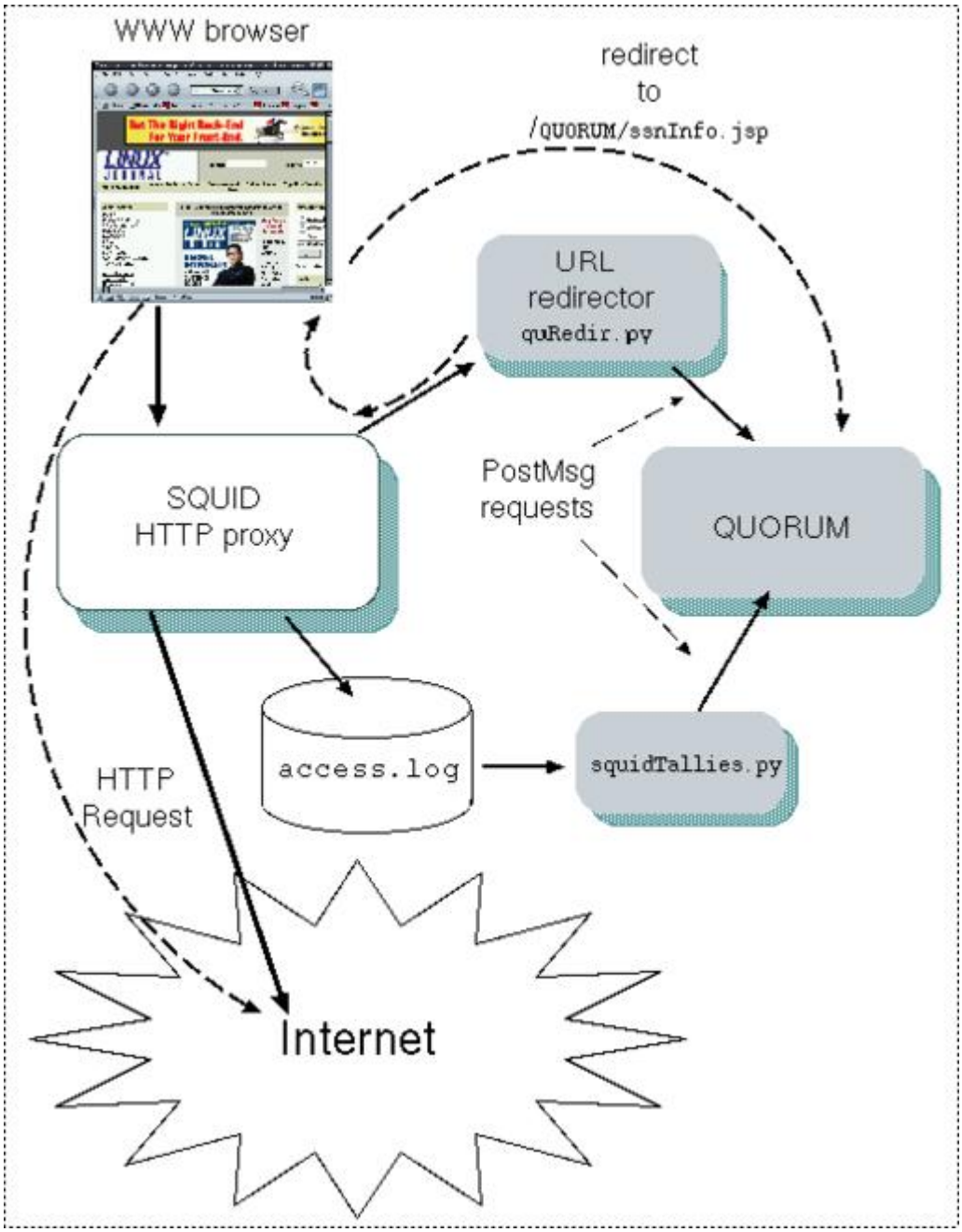


Figure 1. Implementing WWW Usage Quotas

Our bandwidth constraints have since eased with the introduction of a new network for higher education in 2001. We upgraded our internet access to 768Kbps (384Kbps CIR for international traffic), while maintaining our monthly costs at roughly the same level. However, we remain convinced that our quota system is still necessary for maintaining a reasonable QoS for our users.

Squid and URL Redirection

We use NLANR's popular Squid HTTP caching proxy software to provide WWW access for our users. Our firewall ensures that staff and students must use our proxy servers to access the WWW, and we configured the proxies to require login names and passwords for WWW access. With this configuration, the

access.log file generated by Squid has a complete record of all WWW activity for all users. Tallying WWW usage for a given user is straightforward with the information in access.log.

Squid can use external URL redirector programs. When a URL redirector is configured, every URL request to the proxy is sent to the standard input of the URL redirector program, which responds by either telling the proxy to fetch the requested URL or to redirect the request to a different URL. This feature is used by content filtering with programs like SquidGuard where the URL redirector checks each URL request against a database of undesirable URL patterns. Any request for an undesirable URL is redirected to a page telling the user that she may not view the requested URL.

We realised that a URL redirector also could be used to enforce usage quotas. The URL redirector checks the user's quota, and if it has been exceeded, it would redirect her to a page telling her she was out of credit. While simple in principle, this has to happen quickly or it impacts the overall performance of the proxy server. In our implementation, the URL redirector usually responds in two milliseconds when the URL request is permitted.

QUORUM Server

QUORUM (QUOta-based Resource Usage Manager) is a server that provides a simple message API with two basic message types: tallies (bill a particular account/session for an item) and queries (ask the server if a given account/session is still in credit).

We use the QUORUM server to manage the tallies of all URLs retrieved by the proxy as well as the current credit balances of user accounts and also to track user sessions, which correspond to periods of uninterrupted WWW usage by a user. The server keeps a cache of all active account and session tallies and credit balances. Updated items in these caches periodically are written to a database. The server has a WWW interface for displaying account balances, user sessions and other administrative and debugging information.

An early version of the server was written in C, using various CGI programs to provide the WWW interface. However, this approach was abandoned when we discovered Java servlets. The QUORUM server is written as a collection of Java servlets and JSPs, with JDBC for the back-end database connectivity. We currently run it under Caucho's Resin Servlet/JSP container. We use MySQL for the back-end database with Resin's built-in JDBC classes for MySQL access and database connection pooling.

Java servlets are a natural choice for implementing an application like QUORUM. The server contains several persistent shared data structures for

user sessions, per-session tallies and caches of account tallies and credit balances. Since servlets are simply Java classes that are part of a container application that implements a WWW server, it is simple to create these persistent objects as part of the server, as well as background threads that check for idle sessions and periodically flush changed data to the accounting database.

How It Works

When a user requests a URL, her browser sends the URL request to the Squid proxy. Squid then sends this URL request information to the URL redirector. The URL redirector sends a querySsn message to the QUORUM server to ask if the user has a current QUORUM session and if the user's account is still in credit. If everything is okay, then the URL redirector replies to Squid with a blank line telling Squid to fetch the requested URL.

If the user does not have a current session, or the querySsn response indicates that the user's account has no credit, the browser is redirected to a JSP, ssnInfo.jsp, in the QUORUM server that shows the user's current account usage and available credit and asks her to start a QUORUM session by clicking on a link to another JSP, beginSsn.jsp. If the user's account is in credit, this JSP creates a QUORUM session for the user.

The URL redirector actually sends a redirect to a servlet that sends another redirect to ssnInfo.jsp. The URL redirector passes the user name and IP address of the user as a parameter in the request to the servlet:

```
/QUORUM/servlet/Redirect?ssn_id=uname@ipaddr
```

The servlet saves the ssn_id in the HTTP session associated with the client's browser. This lets all the other JSPs use an HTTP session variable to retrieve the user's account and session information.

Usage accounting depends on the information in Squid's access.log file. After a URL has been fetched by Squid, it appends a line to the access.log file, containing the URL fetched, the size of the object in bytes, the user ID of the requester and the IP address of the workstation making the request. A Python script, squidTallies.py, reads the tail of access.log and generates tallySsnItem request messages to the QUORUM server. The QUORUM server processes these tally messages and keeps running usage tallies for user accounts and sessions, as well as credit balances for all active user accounts.

Account balances, usage tallies and logs of user sessions are written to a MySQL database. A background thread in the QUORUM server periodically walks through the caches of session tallies, account tallies and credit balances

and flushes any changed data to the database. This enables the server to keep reasonably up-to-date information in the database without overwhelming the database with multiple updates for every single URL fetched.

The PostMsg Servlet

QUORUM request messages are simple text messages terminated by a newline. For example:

```
ref000001 querySsn ssn_id=soren@1.2.3.4 ccode=11000
```

is a request message asking the server if the user soren at IP address 1.2.3.4 has a current QUORUM session, and if the user is still in credit for “cost code” 11000. The first field of all request messages is a user reference that is echoed back to the client in the response message, which allows multithreaded clients to match responses to request messages.

As a J2EE application, you might expect a hipper message transport such as SOAP over HTTP. We considered encoding the requests as individual HTTP GET requests to servlets that would implement the various QUORUM API messages. This would have been a cleaner fit with the servlet framework, but our tests indicated that the HTTP overhead would be too large for our application. The peak demand from our student labs is 20-30 URLs per second. Since each URL request would correspond to two QUORUM request messages (one querySsn request and one tallySsnItem request), the server would need to handle 60 requests per second comfortably. We use a single QUORUM server to handle both our staff and student proxies, so we need to handle at least 100-150 requests per second. We could not achieve these rates using a separate HTTP request for each QUORUM request message, even making back-to-back requests over a single persistent HTTP connection.

Instead, a single servlet, PostMsg, handles all QUORUM request messages from clients. A client application makes an HTTP POST request and then sends QUORUM request messages on the TCP connection of the POST request, as if it was uploading a file or other data in the request. The PostMsg servlet reads request messages, one per line, from the ServletInputStream with the readLine() method of the input stream. It processes the request messages and sends responses back to the client on the ServletOutputStream. After sending the response, it calls the flushBuffer() method on the ServletHTTPResponse object to ensure the response is sent to the client. Using this technique, a single QUORUM server can handle several thousand requests per second on a single client connection.

There are two minor problems with this technique. First, the servlet container will close any HTTP connection that is idle for more than a set time, typically 30

seconds or so. This means that a client will be disconnected when it doesn't send any requests for a while. The client will then have to reconnect and send another HTTP POST request in order to send more QUORUM request messages. We wrote a Python class that provides a UNIX pipe for sending requests and reading responses. The class establishes the HTTP connection on demand and sends the HTTP POST request. An advantage of the connect-on-demand technique is that the message transport is very robust—idle connections don't hang around and there aren't problems with clients getting stuck, thinking they have a connection that the server thinks is closed.

The other problem is that HTTP/1.1 requires that a POST request has a Content-length header in both the request and the response. While we found that Tomcat ignored this restriction, Resin did not, and it closed the connection when the number of bytes sent by either client or server exceeded the Content-length value. Our workaround was to set a very large value in the Content-length header and arrange for the client to close the connection well before that number of bytes was sent.

Performance of the URL redirector is critical for the proxy as each URL request is delayed until the URL redirector responds. QUORUM's URL redirector keeps a cache of querySsn replies for all recent sessions. This makes the redirector very fast for URL requests once a session has been established and is in credit, as the URL redirector responds based on the most recent reply message in the cache. The response time in this case is typically two milliseconds or less. Even if there is a cached reply, the redirector always sends a querySsn request to the QUORUM server. The reply message is used to update the reply cache, so the cache always contains up-to-date information about the status of user sessions and credit balances.

When the redirector doesn't have a cached reply message, or if the cached reply is negative (user out of credit), then the redirector sends a querySsn request and blocks the reply to the proxy server until the response from the QUORUM server has been received. In this case, the redirector may take from 20 to 50 milliseconds to respond.

Squid can be configured to start up several copies of the URL redirector so that URL redirection requests can be processed in parallel. The QUORUM URL redirector is multithreaded so that Squid can make several simultaneous URL requests. Squid actually talks to several copies of a small stub program that forwards the URL redirector requests to the URL redirector process over unix-domain socket connections. This allows the URL redirector to share the same cache of querySsn replies for all the requests.

Deployment and Administration

Painless administration of user accounts is crucial to the successful adoption of this system. In our deployment of QUORUM, students must have a usage quota for WWW access. Staff usage is currently tallied by the QUORUM server, but staff members are not yet subject to quotas. (Tracking staff usage with QUORUM has been invaluable in cracking down on the use of stolen staff accounts by students, however.)

For internet usage related to a particular course, we credit all students in the course with a specified amount of usage credit, using our database of student registration information. We do this at the start of every term for all courses where the lecturers have requested internet access.

In the past, we found that there were continual cases of students who had registered late, whose registration was not in our database for some other reason or who simply needed additional usage quota. We now have additional discretionary accounts for some departments. Nominated lecturers can transfer funds from their departmental accounts to individual student accounts. This relieves our department of an administrative burden and gives the departments some discretion for the management of student accounts.

For students who require additional usage over and above the quota they are issued, we will introduce a prepaid voucher system. This system is intentionally reminiscent of the voucher systems for prepaid cell phones, which are very popular with our students. Students will be able to buy a voucher from the university bookshop that has a secret access number. The voucher can be redeemed by typing this number into a WWW form and submitting it. The MD5 hash of the number entered is compared to hash values stored in a database table, and if the hashed value matches one of these, the student's account is credited with the amount shown on the voucher. We have had initial discussions with the suppliers of our new campus point-of-sale (PoS) system to see if we can sell our internet quota vouchers using a system similar to the one currently used to sell prepaid cell-phone vouchers at till points.

The voucher system is an important part of our goal to offer a flexible internet service to our users, subsidize internet use for academic purposes and recover the costs of other usage. We currently are testing the voucher system. There is pressure from the students to introduce the voucher system quickly, but we are being very cautious about the rollout of the system as we this will be a service that students are paying for directly. It is important that the system is well established and well understood by our student users so they can be completely comfortable with what they are paying for.

Future Directions

From the outset, QUORUM was designed for applications beyond WWW usage accounting and quota management. Any service that generates a log file of usage information can be parsed by a script that sends tally request messages to the QUORUM server. In the near future, we will add accounting for e-mail usage so that mail messages with large attachments are billed against the sender's usage quota. Gratuitous e-mail usage (sending that cool AVI to all your friends) has been a recurring problem at our site. In the future, our users will have to decide if sending a large e-mail attachment now might mean having to pay for WWW access later.

Usage accounting in QUORUM is structured into cost codes. Currently, WWW usage is broken down into international and national traffic, which are charged at different rates, and Squid cache hits, which are not charged for. The breakdown of charges is hierarchical, where internet usage may be split into WWW and e-mail access, each of which is split into further categories. Eventually, we plan to have additional usage cost codes for network printing, dial-up access and direct TCP/IP traffic through our firewall. The hierarchical structure of the cost codes means that we can have one quota for all services or that specific quota credits apply to only parts of the cost-code hierarchy. For example, we could decide that the quota credits issued to students in a course apply only to internet access, but prepaid voucher credits could be used for network printing as well.

We are trying to utilise our internet bandwidth more effectively by encouraging our users to do so. Our charging structure benefits users who download large files from mirror sites within South Africa rather than from overseas. It is cheaper for us to purchase guaranteed bandwidth for national destinations than for overseas ones. Similarly, we offer after-hour discounts to encourage users to download large files in the evenings when our internet access is less congested.

Another intriguing idea is to integrate a messaging/bulletin system into QUORUM, a sort of intranet instant messaging. We currently can see which users are browsing the WWW. QUORUM could be extended so messages could be sent to a user or group of users. When one of the recipients next requests a URL, the QUORUM server could redirect the user to a page that shows the message/bulletin.

Conclusion

We have developed a flexible accounting and quota management system that works with our existing Linux and OSS-based internet services platform. This system will play an important role in offering a well-managed internet service

on our campus, both from the control it offers over usage and the information it provides about the demand for services.

Soren Aalto (soren@pan.uzulu.ac.za) works in the Networking Services Unit at the University of Zululand, where he divides his time between system administration, WWW application development and converting others to the joys of Linux. His wife lectures at the university and they have two children, three cats, a dog and a varying number of tropical fish.

Archive Index Issue Table of Contents

Advanced search

2002 Readers' Choice Awards

Heather Mead

Issue #103, November 2002

The votes are in, and—big surprise—they're not much different from last year.

Judging by the results of this year's Readers' Choice Awards, things in the world of Linux are holding steady. Almost 6,000 voters participated this year, casting ballots in such categories as favorite distribution, office suite and development tool—a total of 25 categories. All in all, while a few nominees switched places, this year's results aren't that much different from 2001's results. Perhaps this represents a solidification of the market? Have the best products made it to the top of the heap and are only improving their offerings? Or will 2003 bring something revolutionary—assuming politics and Hollywood haven't outlawed open source and the Internet completely by then, of course.

Favorite Distribution

1. Mandrake Linux

2. Red Hat

3. Debian

The distributions in the top three spots have been the same for several years now, but the lineup this year is different. For the first time in three years, Red Hat isn't in the number one position, having been outvoted by Mandrake Linux. The most popular write-in distribution was Russia's ASPLinux. For some, it doesn't seem to matter what the distribution is, as one voter wrote, “I'm addicted to the installation process, so I switch all the time.” Yeah, that's good fun.

Favorite Graphics Program

1. The GIMP

2. CorelDraw

3. ImageMagick

I swear, I don't know why we even bother with some of these categories—like favorite graphics program. In the entire time we've had this category (six years), nothing has come close, ever, to beating The GIMP. The GIMP: good for graphics, bad for contests.

Favorite Word Processor

1. OpenOffice

2. StarOffice

3. AbiWord

Our voters certainly won't complain about getting a free, featureful processor (or office suite, for that matter), especially when the original StarOffice it's based on is no longer free of charge. StarOffice drops to second place (OpenOffice received over a 1,000 more votes), and the love for AbiWord keeps spreading.

Favorite Text Editor

1. Vim

2. vi (and vi clones)

3. GNU Emacs

Are Vim users “a rabid pack of fanatical lunatics”? The Vim web site denies it, so we won't push the matter. Vim is simply a wonderful tool and, apparently, much better than vi, which received half as many votes. Now if you want to talk fanatical, look no further than the users who made Emacs the third-place editor; those guys are nuts. Over on the write-in side, Kate is proving popular enough to be on our official list next year.

Favorite Desktop Environment

1. KDE

2. GNOME

3. Window Maker

The top three picks this year were the same as those from last year, in the same order, with about the same percentage of votes for each (twice as many votes for KDE as for GNOME). Looking beyond the mainstream, the most popular write-in was fluxbox. Quite a few voters also like the minimalist approach to window management employed by Ion.

Favorite Office Suite

1. OpenOffice

2. StarOffice

3. KOffice

The OpenOffice love grows even stronger in the office suite category, beating StarOffice by almost 2,000 votes. A lot of write-ins said they use Microsoft Office because that's “what the office uses” or for compatibility. The next year could change all of that based on rumors of what's being planned for Linux on the desktop.

Favorite Programming Language

1. C

2. C++

3. Perl

C++ kicked Perl out of the second-favorite position this year, and only 17 votes kept C++ out of the top spot. In its first year on the “official” list, Kylix/Object Pascal came in fourth. Following that was a close vote spread between PHP, Java and Python, in that order. One quite reasonable voter wrote in that he uses “whichever is best for the project”. And to the voter who felt bad about preferring bash shell scripting, don't worry, you're not alone.

Favorite Development Tool

1. GCC

2. Kylix

3. Emacs

GCC won by a country mile again this year, but Kylix made a strong second-place showing in this category, collecting two-thirds as many votes as GCC. Fans

of the ever-flexible Emacs kept it in the top three again this year. The write-in list for this category was extensive and included Vim, Visual Works Smalltalk, Visual SlickEdit and mod_perl.

Favorite Shell

1. bash

2. tcsh

3. zsh

The printout of votes collected in this category is always the shortest, tidiest of the bunch. It's not that people don't vote in the favorite shell category—over 5,000 people did—it's that everybody's so quiet about it. So what can be said about bash? It's dependable, flexible, extendable, hardworking and 83% of voters chose it as their favorite.

Favorite *Linux Journal* Column

1. Cooking with Linux

2. Kernel Korner

3. Paranoid Penguin

Marcel Gagné will be very happy to learn that he's your favorite for the second year in a row. Maybe poor François can have a glass of wine to celebrate instead of running to the cellar all night. To those of you who wrote in that all the articles are your favorite, thank you; the checks are in the mail.

Favorite Processor Architecture

1. AMD Athlon

2. Intel Pentiums

3. AMD Duron

Athlon and the various Pentiums were the big winners this year. Combined, the two processors received 74% of the total votes, with about two-thirds of that percentage going to Athlon. Not too many write-ins for this category, but Itanium, Power4 and Zilog Z-8000 (!) all made appearances.

Favorite Communications Board

1. Cyclades

2. Digi International

3. Sangoma

Okay, guys, we explained this category last year, but once more: the communications board category includes things like internal WAN router cards that let servers act as WAN routers and multiport serial cards to connect printers, point-of-sale devices and the like. Of the votes collected, Cyclades is the favorite for another year.

Favorite Database

1. MySQL

2. PostgreSQL

3. InterBase

By a two-to-one margin, MySQL is the voters' favorite again this year. MySQL won the *LJ* Editors' Choice Award this year too. Last year's third-place winner, Oracle, slipped to fourth place this time, replaced by InterBase. The write-in favorite is Firebird, a commercially independent relational database based on InterBase source code. To the voter who asked—no, a “haphazard arrangement of XML files” does not count.

Favorite Backup Utility

1. tar

2. Amanda

3. Arkeia

tar won by a landslide again this year, receiving just under 2,000 more votes than its closest competitor and 90% of the total votes in the category. **rsync** is the big write-in favorite. Of course, the point is moot because “real men don't need backups”—right? Well, maybe only the guy who followed up that comment with the admission he'd deleted his hard drive twice.

Favorite Programming Beverage

1. Coffee

2. Water

3. Tea

The write-ins for this category are always fun, because we get to catch up on all the new sodas and coffee drinks available, especially those available abroad. Sometimes the emotions run as high here as they do in the favorite distribution category. What we learned: Coca-Cola *cannot* be lumped in with the more general Soda category; some of you actually like Vanilla Coke; they still make Afri Cola; Hi-C isn't only for kids; and Swedish coffee kicks wussy-American coffee's butt. Between caffeine and sugar, you people are wired to the gills.

Favorite Linux Game

1. *Quake 3*

2. *Tux Racer*

3. *Freeciv*

We collected 3,514 total votes in the favorite game category, and the first-place winner, *Quake 3*, only received 473 votes. Do you know what that means? It means a lot of games are out there, and each one is somebody's favorite. Among write-ins, *Frozen Bubble* and *Return to Castle Wolfenstein* are the most popular.

Favorite Web Browser

1. Mozilla

2. Galeon

3. Konqueror

Last year's winner, Netscape, fell to fourth and fifth place this year (we split it out into Netscape 4.x and Netscape 6.x), as Mozilla overwhelmingly claimed the title of favorite web browser. Galeon, the GNOME browser based on the Mozilla rendering engine, picked up the slack and rushed in to second place. Thankfully, the number of Internet Explorer write-ins dropped significantly.

Favorite Web Site

1. Slashdot

2. LinuxFR

3. Freshmeat.net

Maybe all the Francophiles who picked Cooking with Linux as their favorite *LJ* column are also fans of LinuxFR.org, Da Linux French Page, which took second place this year. As always, Slashdot's endless stream of updates on open source, digital rights and new Mandrake user Wil “Ensign Wesley Crusher Must Die” Wheaton makes it the most popular web site one more time. PCLinuxonline.com is the most popular write-in vote.

Favorite E-Mail Client

1. KMail

2. Evolution

3. Mozilla

In a monster shake-up, the GUI mailers swept the top three for the first time this year. The highest-ranked text-based mailer, mutt, fell to number 4. Last year, Netscape won; this year, it's in sixth place. It looks like the release of KDE 3.0 encouraged some switching to KMail, the only client from last year's top three to return this year. Ximian's Evolution entered the category in second place.

Favorite Instant-Messaging Client

1. gaim

2. Licq

3. Xchat

This year's favorite IM client is gaim, which didn't make the top of last year's list nor did it make much of an appearance in last year's write-in votes. Trailing not too far behind with less than 50 votes is Licq, a multithreaded ICQ clone with a Qt/KDE interface.

Favorite Distributed File-Sharing System

1. Gnutella

2. audiogalaxy

3. MORPHEUS

Remember the good ol' days when you could go to Napster and download every last b-side of some obscure '60s Britpop band or see just how many people had covered Dolly Parton's song “Jolene”? Or were those the bad ol' days? Well, the music industry still is trying to figure out what to do with music on the Internet (the Musicnet and Pressplay services didn't even show up among the write-ins), but Gnutella is going strong, receiving 800 more votes than its nearest competitor.

Most Indispensable Linux Book

1. *Linux in a Nutshell* by Ellen Siever

2. *Running Linux* by Matt Welsh, et al

3. *Linux System Administration* by Vicki Stanfield

These three titles have been at the top of their class pretty much since we started these awards. It's amazing to see how many titles in Linux and Linux-related areas are published every year. Judging by the ever-increasing list of write-in votes, everybody's found an indispensable book to call their own—except for those of you who see no need for books because “everything is available on the Web and in man pages.”

Favorite Ad-Filtering Tool

1. Junkbuster

2. Homemade

3. SquidGuard

The top three vote-getters this year are an exact repeat of last year's. Not too many voters responded to this particular category, so perhaps you've got a quick trigger finger and zap those guys, or you suffer through them. Three write-ins even said it was morally wrong to filter ads, but they're in a very small minority. According to write-ins, a lot of you at home are using Mozilla's pop-up blocking option.

Favorite Embedded Distribution

1. MontaVista Linux

2. Qtopia

3. Lineo

This is the first year that favorite embedded distribution was included in the survey. Not a lot of votes were collected, but MontaVista had one-third of the total votes, making it the favorite. A few of you are building or have built your own.

Favorite Audio Tool

1. xmms

2. Vorbis Tools

3. mpg123

By far, xmms is the winner of the favorite audio tool award, and perhaps by the biggest margin of any category—2,300 votes between it and the second-place Vorbis Tools. Noatun, the KDE media player, is the favorite among write-ins.

Heather Mead is senior editor of *Linux Journal*. She likes depressing movies, expensive shoes and well-mixed cocktails.

Archive Index Issue Table of Contents

Advanced search

Using the Kernel Security Module Interface

Greg Kroah-Hartman

Issue #103, November 2002

Greg shows how to create a simple kernel module that uses the LSM framework.

At the 2001 Linux Kernel Summit, NSA developers presented their work on Security-Enhanced Linux (SELinux) and emphasized the need for enhanced security support in the main Linux kernel. In the ensuing discussion, a consensus was reached that a general access-control framework for the Linux kernel was needed. This approach would allow different security models to work without modifying the main kernel code.

Out of this discussion grew the Linux Security Module Project (LSM). A number of developers worked together to create a framework of kernel hooks that would allow many security models to work as loadable kernel modules. A detailed description of the design of this general-purpose system was published in a paper at the 2002 USENIX Security Conference (lsm.immunix.org/docs/lsm-usenix-2002/html/) and a technical description of how the LSM interface works was presented in a paper at the 2002 Ottawa Linux Symposium (www.linux.org.uk/~ajh/ols2002_proceedings.pdf.gz).

During the 2002 Linux Kernel Summit, the technical description of the project was presented, and the first portion of the LSM framework appeared in the 2.5.29 kernel release. Further kernel releases contained more portions of the LSM framework, and hopefully the entire patch will be included by the time of the 2.6.0 release.

This article does not attempt to describe how the LSM framework works or the design decisions that were made in creating it; the previously mentioned references do an excellent job of that. Instead, this article shows how easy it is to create a simple kernel module that uses the LSM framework.

Root Plug

Our example uses the 2.5.31 kernel release, which contains enough of the LSM interface for us to create a useful module. In our module, we want to prevent any programs with the group ID of 0 (root) from running if a specific USB device is not plugged in to the machine at that moment. This provides us with a simple way of preventing root exploits from running on our machine, or for new users to log in when we are not present.

This example creates a kernel module called root_plug, which is available as a patch against a clean 2.5.31 kernel tree from the *Linux Journal* FTP site [ftp.linuxjournal.com/pub/lj/listings/issue103/6279.tgz].

For a description of UNIX systems that handle the user and group ID values and how they interact with the setuid class of system calls, see the excellent paper by Hao Chen, David Wagner and Drew Dean entitled “Setuid Demystified”, which was presented at the 2002 USENIX Security conference (www.cs.berkeley.edu/~daw/papers/setuid-usenix02.ps).

The LSM interface is four simple functions:

```
int register_security
    (struct security_operations *ops);
int unregister_security
    (struct security_operations *ops);
int mod_reg_security (const char *name,
                  struct security_operations *ops);
int mod_unreg_security  (const char *name,
                  struct security_operations *ops);
```

A security module registers a set of security_operations function callbacks with the kernel by calling the function register_security(). If that fails, it means that some other security module probably has been loaded already, so the mod_reg_security() function is called in an attempt to register with this security module. This can be seen in the following code:

```
/* register ourselves with the security framework */
if (register_security (&rootplug_security_ops)) {
   printk (KERN_INFO
       "Failure registering Root Plug module "
       "with the kernel\n");
   /* try registering with primary module */
   if (mod_reg_security (MY_NAME,
                         &rootplug_security_ops)) {
       printk (KERN_INFO "Failure registering "
               "Root Plug module with primary "
               "security module.\n");
       return -EINVAL;
   }
   secondary = 1;
}
```

When the module wants to unload itself, the reverse process must happen. If we used mod_reg_security() to register ourselves, the mod_unreg_security()

function should be called, otherwise the unregister_security() function is the proper thing to call. The following code shows this logic:

```
/* remove ourselves from the security framework */
if (secondary) {
   if (mod_unreg_security (MY_NAME,
                           &rootplug_security_ops))
      printk (KERN_INFO
              "Failure unregistering Root Plug "
              " module with primary module.\n");
} else {
   if (unregister_security (
       &rootplug_security_ops)) {
      printk (KERN_INFO "Failure unregistering "
              "Root Plug module with the kernel\n");
   }
}
```

The rootplug_security_ops is a large structure of function pointers that are called when various events happen in the kernel. This includes such things as whenever an inode is accessed, a module is loaded or a task is created. As of the 2.5.31 kernel, there were 88 different function pointers needed. The majority of these functions not needed by most security models, but they must be implemented, or the kernel will not work properly. If a security module does not need to do anything for a specific hook, a “good” value needs to be returned to the kernel. An example of this can be seen in the following function:

```
static int rootplug_file_permission
    (struct file *file, int mask)
{
    return 0;
}
```

This function is called whenever the kernel wants to determine if a specific file can be accessed at this moment in time. A security module can look at the file, check whether the current user has proper authority and possibly refuse to grant it.

What Hook to Use

For our example module, we want to be able to stop a new program from being run if our USB device is not present. This can be done by using the bprm_check_security hook. This function is called when the execve system call is made, right before the kernel tries to start up the task. If an error value is returned from this function, the task will not start. Here is our hook function:

```
static int rootplug_bprm_check_security
    (struct linux_binprm *bprm)
{
    if (bprm->e_gid == 0)
        if (find_usb_device() != 0)
            return -EPERM;
    return 0;
}
```

This function checks the value of the effective group ID at which the program is to be run. If it is zero, the function find_usb_device() is called. If the USB device

is not found in the system, -EPERM is returned, which prevents the task from starting.

Finding a USB Device

The find_usb_device() function simply goes through all of the USB devices in the system and sees if the device specified by the user is present. The USB devices are kept in a tree, starting at the root hub device. The different root hubs are kept in a list of buses. These buses are checked in order in the find_usb_device() function:

```
static int find_usb_device (void)
{
    struct list_head *buslist;
    struct usb_bus *bus;
    int retval = -ENODEV;
    down (&usb_bus_list_lock);
    for (buslist = usb_bus_list.next;
         buslist != &usb_bus_list;
         buslist = buslist->next) {
        bus = container_of (buslist,
                            struct usb_bus,
                            bus_list);
        retval = match_device(bus->root_hub);
        if (retval == 0)
            goto exit;
    }
exit:
    up (&usb_bus_list_lock);
    return retval;
}
```

The match_device() function looks at the device passed to it. If it matches the expected device, then it returns success. Otherwise, it looks at the children of this device, calling itself recursively:

```
static int match_device (struct usb_device *dev)
{
   int retval = -ENODEV;
   int child;
   /* see if this device matches */
   if ((dev->descriptor.idVendor == vendor_id) &&
       (dev->descriptor.idProduct == product_id)) {
       /* we found the device! */
       retval = 0;
       goto exit;
   }
   /* look at all of the children of this device */
   for (child = 0; child < dev->maxchild; ++child) {
       if (dev->children[child]) {
           retval =
               match_device (dev->children[child]);
           if (retval == 0)
               goto exit;
       }
   }
exit:
   return retval;
}
```

Specifying a USB Device

Because every user has different types of USB devices, specifying the device to look for must be done in a simple manner. All USB devices have a specific vendor and product ID. You can see these values by using the lsusb or usbview program when there are some USB devices plugged in to your system. This information also is shown in the /proc/bus/usb/devices file, in the lines starting with "P:". See the Documentation/usb/proc_usb_info.txt file for more information on how the data in this file is presented.

The match_device() function looks to see if the value of the specific device matches the vendor_id and product_id variables. These variables are defined in the code as:

```
static int vendor_id = 0x0557;
static int product_id = 0x2008;
MODULE_PARM(vendor_id, "h");
MODULE_PARM_DESC(vendor_id,
            "USB Vendor ID of device to look for");
MODULE_PARM(product_id, "h");
MODULE_PARM_DESC(product_id,
           "USB Product ID of device to look for");
```

This allows the module to be loaded with the vendor and product ID specified on the command line. For example, if you want to specify a USB mouse with vendor ID of 0x04b4 and product ID of 0x0001, the module would be loaded with:

```
modprobe root_plug vendor_id=0x04b4 \
product_id=0x0001
```

If no vendor or product ID is specified on the module load command line, the code defaults to looking for a generic USB to serial converter with a vendor ID of 0x0557 and a product ID of 0x2008.

Building the Module

Finally, we need to add our module to the kernel build process. This is done by adding the following line to the security/Config.in file:

```
tristate 'Root Plug Support'
CONFIG_SECURITY_ROOTPLUG
```

And the following line to the security/Makefile file:

```
obj-$(CONFIG_SECURITY_ROOTPLUG)    += root_plug.o
```

These changes allow the user to select this kernel module either to be built into the kernel directly or as a module. Run your favorite *config option to select the "Root Plug Support" (**make oldconfig** will work nicely here, as only the new

option will be asked about if you already have a working .config file set up for your kernel). Then build the kernel as usual.

After your kernel is built and running, load the root_plug module by typing (as root):

```
modprobe root_plug vendor_id=<YOUR_VENDOR_ID> \
product_id=<YOUR_PRODUCT_ID>
```

Now try to run a program as root with your specified USB device plugged in to your system, and then try it without. With the module loaded, and the device removed, the following error happens on my machine:

```
$ sudo ls
sudo: unable to exec
/bin/ls: Operation not permitted
```

Plug the device back in, and things should work just fine.

But Is It Secure?

This example shows how powerful and simple the LSM interface can be. With one hook, any program with the root group ID is prevented from running unless a device is physically present in the system.

Using this code, if the device is not present, users are not allowed to log in to the console, as mingetty traditionally runs as root. But users can log in through SSH as normal users, as sshd already was running before the device was removed. Web pages also can be served, and other services that do not run as root (your mail server, database server, etc.) also will function properly. If one of these server programs were broken into, and they tried to spawn a root shell, that root shell would not be allowed to run.

This module does not prevent any program already running as root from cloning itself, or keep a program from trying to change the privileges that are currently assigned to it. To check for these things, the task_* functions in the security_operations structure should be used. The implementation of these functions will be much like the bprm_check_security function, but the parameters passed to the function will be different, so the egid will need to be determined differently.

There are probably other methods of taking an existing running program and spawning a root process that this module does not catch. Please do not use it in a production environment, but rather as a learning exercise for how to create other LSM example code.

Acknowledgements

I would like to thank Chris Wright, Stephen Smalley, James Morris and all of the other programmers who helped create the LSM interface and get it accepted into the main kernel tree. Due to their hard work, Linux now has a flexible security model that will give everyday users the ability to have access to different security models with little effort. I also would like to thank Alan Cox for the initial idea that spawned this example.

For more information about the LSM Project, the development mailing list, documentation and patches for different kernel versions, please see the web site at lsm.immunix.org.

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

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Controlling Creatures with Linux

Steve Rosenbluth

Michael Babcock

David Barrington Holt

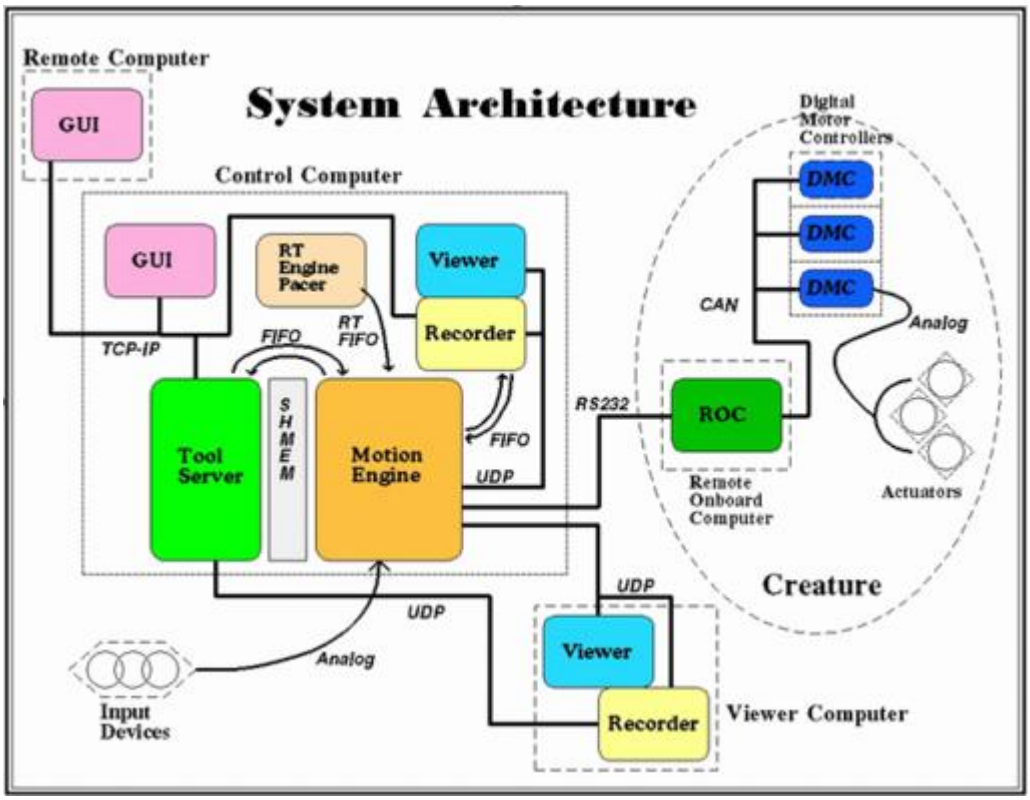
Issue #103, November 2002

How embedded Linux satisfies the various real-time needs of The Jim Henson Company's animatronics and 3-D computer graphic puppets.

The Jim Henson Company is well known for creating characters. Low-tech characters like the Muppets don't need much technology, but animatronics, from gerbils to dinosaurs, do need it, not to mention our 3-D computer graphic puppets. Performing live, in real time, so they can interact with human actors and be captured on film, these characters have a curious set of needs from a technology perspective.

One of Jim Henson's original performance goals was that one person should be in command of each character, bringing a spontaneity and personality harder to achieve in a “performance by committee” (where several people perform a puppet together). The fascinating thing about a creature that achieves this goal is that people forget who or what is controlling it and simply interact with it. Actors and audience alike start conversing with a dog or a frog or a snowman as if it were human.

With the proliferation of servo motor technology in animatronic puppets in the early 1980s, managing increasing numbers of servos became a challenge, so computerized control systems were designed. During the last 15 years, several generations of control systems have been developed at the Jim Henson Creature Shop, including a version that won a Technical Achievement Academy Award in 1992. The latest Henson Performance Control System (HPCS) encompasses the best features of previous systems, while adding new technology available only with today's hardware and computing environments such as Linux.



Schematic Diagram of PCS/HDPS System

This system was begun under the guidance of Computer/Electronics Supervisor Jeff Forbes in early 1998. We had a vision that one system on a standard architecture could service all the company's needs. Steve Rosenbluth joined the project at that point as the control system designer, and Michael Babcock as the multimedia programmer. Our needs turned out to be rather expansive, and Linux seemed to be the only thing that could do it all without flinching.

The system has to support two back ends: one animatronic and the other computer graphic. So, our puppets are either real-world robots, or virtual puppets made of polygons and pixels. We can handle either separately or both at the same time.

Once the software “set up” of a puppet is in the system, even puppeteers new to the technology can jump in and perform well within hours. Using the input devices is akin to performing a musical instrument. At a certain point, the puppeteer, like the musician, no longer has to think about what he's doing—he just performs.

PCS setup for animatronic puppets, showing input controls and Control System laptop displaying the GUI.

Henson input devices are not motion capture technologies. Motion capture is both directly analogous to the performer and largely is nonprogrammable. In motion capture, a performer's arm simply corresponds to the creature arm, a knee corresponds to a knee, etc. The performer cannot enhance or reprogram these relationships. The Henson input scheme is both non-analogous and user-programmable. Our input devices are abstractions. For example, a puppeteer's index finger might proportionally control the sincerity or sarcasm of a creature's entire face. And a puppeteer can reprogram puppet movement easily between and even during performances. A person in a motion capture suit would be hard pressed to perform an octopus. A person operating our control system could take it in stride.

The Control Computer and Motion Engine

At the core of the system is a Control Computer, running RTLinux, which processes and distributes motion data to puppets. The process that runs motion-mixing algorithms on the Control Computer is called the Motion Engine. Steve Rosenbluth wrote it in C++. Performer movement, coming through input

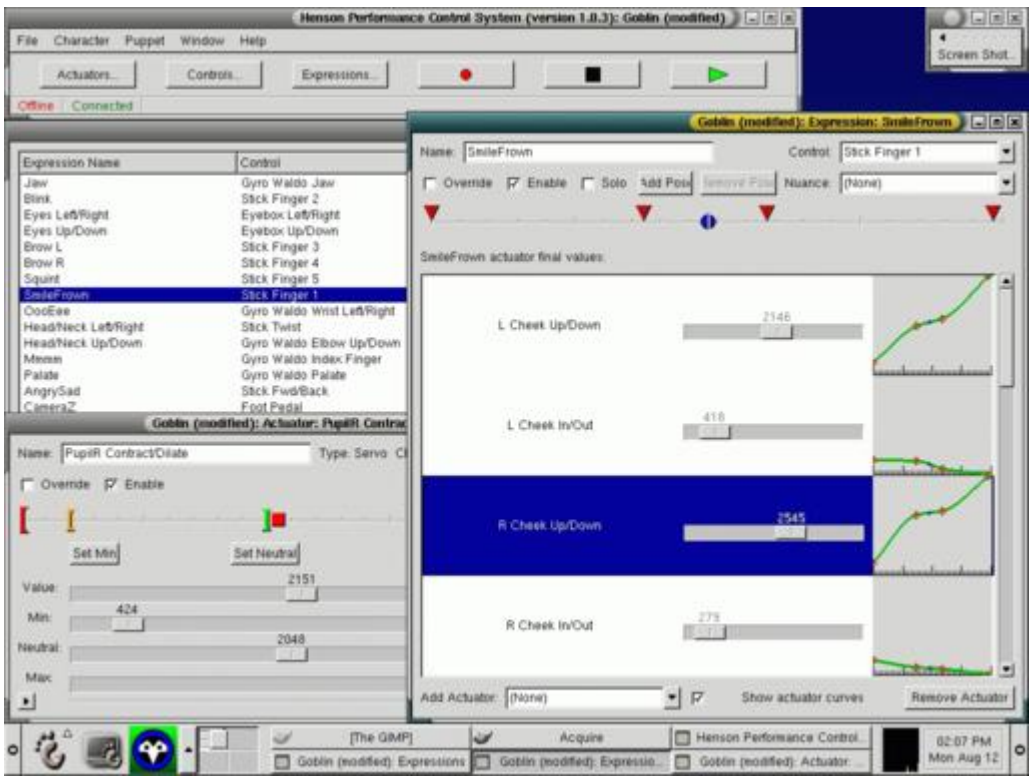
transducers from the outside world, passes through the Motion Engine on its way to networked puppets. Inside the Motion Engine, various algorithms are performed on the live data, resolving final actuator positions. An actuator is like a muscle of the puppet; it could be an electromechanical or hydraulic servo in an animatronic or a “virtual servo” (a mesh deformation) in a computer graphic puppet. Motion-mixing relationships are configurable in software to be one-to-many or many-to-one, and compound mixes can be performed on top of those. Physics, which add effects like weight or smoothing, can be added to performance data as it passes through the Motion Engine.

The Tool Server

Motion-mixing algorithms are served up via software tools provided by a process called the Tool Server. The Tool Server establishes the relationships between live performance data and the algorithms the Motion Engine will perform on it, using direct access to objects and datasets in shared memory. Designed by Michael Babcock, it is, as the name implies, an asynchronous server that speaks with graphical user interface clients that connect to it.

The Graphical User Interface

The GUI is a “not-so-thin” client that connects to the Tool Server via a socket. Michael Babcock wrote the actual application in GTKmm, based on an original high-level interface design by consultant Robert McNally. The client/server architecture allows multiple instances of the user interface to be run either by technicians on other machines or by additional puppeteers.



Screenshot of Sample Window of HDPS GUI

There are times during production when a technician assists puppeteers with administrative tasks, so a networked GUI means the technician can do those tasks without kicking anyone off the system. A custom protocol is spoken between the GUI and Tool Server. The ability for a technician to administer our Control System application remotely is of great value to us, as production locations may be anywhere around the world, and support staff back at The Creature Shop can have complete access for the purposes of debugging or assisting.

Since Linux itself provides mechanisms for remote access to the rest of the operating system, a technician back at The Creature Shop can give us unparalleled support for systems on the road.

As you can see in the schematic diagram, the back end of the control system, beyond the Motion Engine, can vary. We can perform animatronics with embedded processors in a remote onboard computer (ROC) scenario, and we can perform 3-D Computer Graphic puppets with a Viewer back end.

The Remote Onboard Computer

In one 60Hz frame, the motion engine will obtain an analog to digital converter (ADC) snapshot of our physical input devices, execute motion-mixing algorithms based on the character setup and transmit the data to ROC clients.

Our animatronic ROCs are embedded PCs running DR DOS. Steve Rosenbluth wrote the ROC code in “somewhat object-oriented C” for speed, as it is a fairly lean-and-mean piece of code. The ROC protocol allows for multiple devices on the communications link, which is currently RS-232. This aging interface is not as high-bandwidth as modern networking hardware, but it is the easiest to use over fiber, copper and radio. RS-232 is a hard real-time interface with predictable triggering and latency, which we need. We use plastic fiber optics to transmit RS-232 when a creature is tethered and spread spectrum for wireless creatures.

Viewers

The computer graphics Viewer is the software module that renders and displays a computer graphic (CG) puppet on a computer screen. The Viewer represented in the schematic diagram can be one of an array of CG modeling packages or game engines that can render quickly enough to display an OpenGL scene live. Computer graphics puppeteering at the Henson Company was pioneered by Digital Supervisor Hal Bertram in the London Creature Shop in the early 1990s. More recently, some of the PC-based CG applications for

which Michael Babcock has written control system plugins include Discrete's 3-D Studio Max, Side Effects' Houdini, Kaydara's Filmbox and Alias|Wavefront's Maya.

A control system Actuator, in the CG realm, is scalar channel data that can move a 3-D mesh deformation or a blend shape. A UDP network connection from the Control Computer's Motion Engine streams live motion data into the Viewer. Viewers behave as ROC clients from the perspective of the Motion Engine, in that they speak the same ROC protocol.

With dual AMD Athlon machines, we get frame rates above 100FPS, enough to put multiple puppets in one scene. Character Technical Director Jeff Christie helped complete the picture by perfecting a 3-D character setup that gets fast, lifelike performance from our CG models.

The Viewer supports connections from multiple motion engines, which means, for a scene with multiple characters, that each can be performed by its own control system and puppeteer, sort of like a networked game.

The Recorder

Motion and sound can be recorded and played back by a nonlinear multimedia editor called the Recorder, developed by Michael Babcock. The architecture, designed by Michael and Steve, consists of a multithreaded process networked to the Motion Engine via UDP. The Recorder is synchronized to the Motion Engine because it slaves off its output as an ROC client, yet the Recorder also streams stored data back to the Motion Engine for broadcasting to other ROC clients. This networked structure allows each process to have its own timing and I/O requirements, without interfering with the other, as in the Tool Server/ Motion Engine relationship.

Because recorded motion can be cued and played back live, the puppeteer can layer a performance, as one would produce a multitrack audio recording. This is particularly useful for lip-sync scenarios, where the performance of a creature's mouth can be perfected off-line, then played back while the puppeteer performs the rest of the character live.

Dan Helfman contributed a sound recording facility to the SDL, the open-source multimedia API we use in the Recorder.

The Link Supervisor and Puppet Clients

A module within the Motion Engine called the Link Supervisor can broadcast and manage connections with multiple ROC clients, regardless of their network type or implementation. The result is that one puppeteer can control multiple

puppets in multiple mediums. For example, an animatronic cat can be performed at the same time as its computer graphics counterpart. While the body and face of the animatronic is captured on camera, a computer graphics mouth, performed simultaneously, can be viewed live on a monitor or even composited live with the camera tap image on set.

HDPS rackmount setup for 3-D CG characters showing hand controls, GUI screen and Viewer.

This allows each medium to do what it does best. We get the complex lighting and physics of a “real” creature on set, and CG mouth data can be further finessed in postproduction before compositing with the film plate. This live previsualization allows a director to direct truly the creature performance on set, while allowing actors to interact with their creature costars.

Process Architecture

There is a purposeful division between the Motion Engine, the Tool Server, the GUI and the Recorder. Because the more complex multimedia and networking modules require software techniques that might compromise process timing or stability, an architecture was designed by Steve Rosenbluth and Tim McGill that builds a wall around the Motion Engine. The goal was for the Motion Engine to have a minimal amount of complexity so that it keeps running. The Tool Server, expected to grow large and complex, was allowed to go down and restart without affecting the Motion Engine. The architecture also allowed the GUI to come and go without negatively affecting either the Tool Server or Motion Engine, and likewise for the Recorder. To accomplish this, the system was segmented into process modules that communicate via UNIX IPC and networking.

The Tool Server and Motion Engine have a block of System V shared memory in common. This enables immediate updates of critical data objects. They also communicate via two FIFOs for messaging that is sequence-critical. There also are UDP network sockets between the Motion Engine and Recorder, which stream data in soft real time to each other. The Motion Engine is what we call a near-mission-critical application, in that its failure in the field could have negative consequences for us. On-set downtime can cost a film production company many thousands of dollars an hour. It's also the nature of the motion picture industry that actors and crew may be in close contact with the animatronic machinery. It would be a bad thing to have an animatronic dog bite an actor while a technician logs in and restarts applications. That is why there is no GUI or other unnecessary code in the Motion Engine. Given our near-mission-critical requirements, the stability of the Linux operating system itself is a big plus.

The independent process architecture also aided development by allowing individual programmers to write and test more modular, self-contained pieces of code. It gave developers the freedom to use custom, and sometimes cutting-edge, programming techniques safely that weren't necessary or appropriate for the other process modules.

Sequencing and Timing

The timing requirements of the system are varied. The Motion Engine has to have an accurate 60Hz invocation frequency in order to update animatronic motors smoothly. Time-domain jitter in motor position data sent to a puppet adds high-frequency accelerations to that data and could cause a puppet limb to jitter. Our ultimate goal was to have a precise 60Hz Motion Engine frequency, but we planned to arrive there in three architectural iterations, as the system came on-line.

In order to prototype the system, periodic software interrupts were used to invoke the Motion Engine as a handler. Additionally, POSIX.1b SCHED_FIFO prioritization was used to make sure that once the Motion Engine was invoked, it didn't get preempted by the kernel scheduler. This allowed the Motion Engine to run in user space easily, and most importantly, in a debugger. The downsides of alarm handlers are twofold: 1) they can have jitter the magnitude of a timeslice or more, caused by a busy kernel scheduler, and 2) one can't specify their period very accurately, since it is in quantums of kernel timer ticks. We recompiled our kernels to increase the number of timer ticks per second for two reasons: to round the Motion Engine period closer to 1/60Hz and to help insure that lower-priority control system processes got descheduled more frequently, helping them all keep current with the Motion Engine state.

For the second architectural iteration, we created an RTLinux hard real-time periodic thread at a frequency of 60Hz. This thread, because it runs in RTLinux space, is about three orders of magnitude more precise than the kernel scheduler. We refer to this thread as our Hard Real Time Pacer. When it wakes up, it puts a flag into an RTLinux FIFO from RTLinux space, and our Motion Engine, blocking on this FIFO in user space, wakes up when the flag arrives. Although the Motion Engine still relies on the the Linux kernel for invocation, this architecture proved to be more accurate than we anticipated, as I/O latency is of a higher priority in the kernel than signal handling. Typical latencies of the FIFO-blocker pacer are less than 40 microseconds when there is no other heavy system activity, which means that the Motion Engine does have a true 60Hz invocation frequency, as accurate as the CPU timers can provide.

Under heavy load, the kernel may not unblock the Motion Engine on time, so this is not a deterministic hard real-time solution, but it served us well as we started using the system for production work. A dual-processor Athlon motherboard can maintain Motion Engine invocation accuracy while running the GUI, Tool Server and the Viewer, rendering OpenGL scenes in a busy loop!

For the third and final architecture we contracted FSMLabs to add an extension to RTLinux that allows deterministic scheduling of Linux user-space processes. The mechanism, called PSC, allows a sort of “jump” from a RTLinux periodic thread to user space, where we run our Motion Engine, then we fall back to RTLinux and finish. Part of our contract was that the code be donated back to open-source RTLinux for all to use.

Input Devices

The input devices used in the control system are a special combination of linear input potentiometers designed for both ergonomics and flexible use. They are uniquely suited for producing the types of motion needed by a puppeteer. We started with the rather aggressive design goal of running the whole control system on a Linux laptop, for maximum portability “on location”. Only when servicing CGI did we start creating 19" rackmount Control Systems.

Puppeteer at work on the HDPS digital puppeteering station.

One of the challenges of laptops was getting 64 channels of analog data into the machine. No A/D converter (ADC) drivers were available, so Steve Rosenbluth wrote one for the Computer Boards DAS16s/16. To date, no PCMCIA ADC card provides more than 16 channels, so Steve also designed an external analog multiplexer. In order to switch the multiplexer through four banks within one motion control frame (16.6msec) we relied on RTLinux, which gave us the determinacy we needed with sub-20-microsecond accuracy.

While writing low-level ADC drivers can be rewarding, it isn't the best way to spend our available R&D resources. We were elated to find during 2000 that United Electronics Industries offered both Linux and RTLinux drivers for its Powerdaq ADC cards, which we used in our PCI bus systems. Their 64-channel PD2-MF-64-333/16L worked like a charm, and UEI was responsive to our needs as the driver developed.

Animatronic Hardware

Although part of our design philosophy is to use as much off-the-shelf hardware as possible, we do have to design hardware for our specialized needs. Most manufacturers' idea of “small” simply doesn't come close to fitting inside an animatronic hamster.

The Digital Motor Controller (DMC), a specialized piece of hardware about the size of two postage stamps, is essential for animatronics. It distributes dozens of PWM (pulse width modulation) signals to motors inside the puppet. Steve, with the help of Glenn Muravsky, designed an SBC that was based on the Texas

Instruments 320LF2406 motor controller DSP. The parallelism available on this chip allows it to do things one cannot do on a PC bus. Steve also implemented a closed loop PID algorithm on the chip for controlling custom motor servos.

Why Linux?

Linux wasn't originally designed for hard real time at the kernel level, but with the advent of real-time extensions, we found we were able to prioritize our critical tasks while having the general-purpose operating system available.

Preemptive multitasking, memory protection, interprocess communication, networking and multimedia APIs were essential for everything else we wanted to implement that wasn't hard real-time. We found that the RTLinux architecture, where Linux itself runs as the lowest-priority task, gives us the option to add a lot of support applications on the same machine that runs real-time tasks.

The challenge of this project was not any one specific area but orchestrating the many different requirements at once without compromising the needs of any individual piece of the puzzle. Our soft real-time code needed a chance to run; our motion control code needed to run without interruption from lower-priority tasks; we needed to protect critical code segments from less important code segments; we needed to use generic kernel facilities; and we needed to use utility applications as part of our distribution. The facilities of the core operating system and the utilities around it sped our development, as we did not have to roll our own version of everything. But if we ever did run into a jam, we knew we had the source code and thus could be in control of our own destiny. So, ultimately, Linux was the only thing that came to mind that could meet all of our needs.

Incidentally, we've been running these new control systems 24/7 for three years now, and our uptimes average a few months. We frequently have reasons to move machines from one location to another, but we've never had a control system go down in production.

Production Projects

Where can you see the products of our new control system? We cut our teeth in 1999 with “Webisodes” featuring the Muppets on the Henson web site, www.muppetworld.com. We've done a number of live performances at tradeshows, amongst them an interactive CG Kermit the Frog for the keynote speech of SIGGRAPH 2000.

The maiden voyage of the new control system in motion pictures was in 2001 on Walt Disney's *Snowdogs*. The lead Husky in that film, named Demon, had an

animatronic double for difficult shots—try to spot them. As we had hoped, the system performed flawlessly. We also performed an animatronic falcon for *Stuart Little 2* during 2001. Horace D'Fly is a CG character who appears in an upcoming Henson home video feature *Kermit's Swamp Years*. We currently are planning to use the system for a sequel to Warner Brothers' *Cats and Dogs*, plus other feature productions involving animatronics. There is a lot of entertainment industry interest in using our CG back end to produce characters for motion pictures, television shows and video games in the near future. So, we'll see you at the movies!

Steve Rosenbluth is 34 years old, and is married to a wife and a cat. He hails from New York, DC, Florida, Montréal, Belfast and LA. He has been designing robotic creatures and related technology for 13 years. He spent his formative years learning stop-motion animation, sculpture, electronics, programming and small-business development. When not on computers, Steve enjoys biking, hiking and skating but regrets no longer performing guitar or trapeze.

Michael Babcock (michael@kanji.com) grew up in Montana in the mountains among trees and deer. He enjoys playing basketball, hacky-sack and the guitar and listening to The Fall. He has been using Linux since 1992. His programming interests include multi-lingual software (especially Chinese and Japanese) and abstract software design techniques. He has been working at Jim Henson's Creature Shop in Los Angeles for four years on a puppeteering control system, mostly doing network, user interface and 3-D graphics programming.

David Barrington Holt (dbh@la.creatureshop.henson.com) (aka DBH) is the creative supervisor of Jim Henson's Creature Shop in Los Angeles. Born and educated in London, England, he graduated with honours in Industrial Design at the height of the “swinging Sixties”, but his subsequent career has included fashion design, graphic art, photography and engineering model making. He has also trained as a psychotherapist, entertained “too many interests for his own good”, has a Russian wife from St. Petersburg and an 11-year-old son.

Archive Index Issue Table of Contents

Advanced search

Multicast Routing Code in the Linux Kernel

Matteo Pelati

Issue #103, November 2002

Let one packet go to multiple addresses and you can save much bandwidth. That's the promis of IP multicasting, and here's how Linux handles it.

In this article I explain how the Linux kernel manages multicast traffic and how it is possible to interact with it by simply patching some kernel code. Although this is a rather specific topic, it might be useful for anyone interested in multicast routing. If you want to monitor or modify any existing multicast protocol, the information provided below will be useful.

At the University of Milan, we are developing a new protocol, CAMP (Call Admission Multicast Protocol), that uses information provided by the multicast kernel code to make some important decisions. We have to be able to receive notifications of important events, such as JOIN or LEAVE requests. As you probably know, the Linux kernel can act as a multicast router, supporting both versions 1 and 2 of PIM (Protocol Independent Multicast, netweb.usc.edu/pim). All the MFC (Multicast Forwarding Cache) update operations are served completely by an external user-mode process interacting with the kernel. In this article, we explain how the kernel manages messages sent by the user-mode dæmon in order to update the MFC. After this brief introduction we describe our hook implementation in more detail. Figure 1 shows the topology of our testing network. As you can see, SNOOPY is acting as a multicast router running PIMd (version 2.1.0-alpha 29.9) on top of Linux kernel 2.4.18.

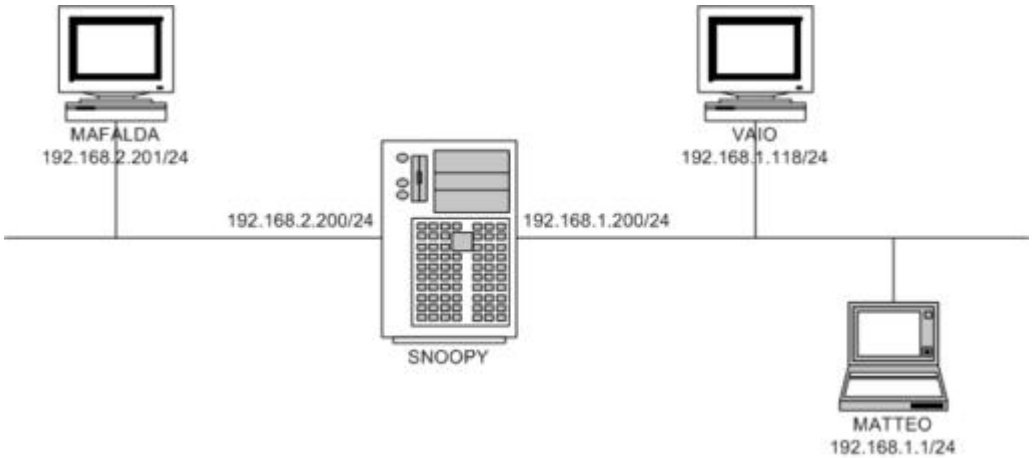


Figure 1. Test Network Topology

All the multicast-related code of the kernel is located in two files: ipmr.c (net/ipv4/ipmr.c) and mroute.h (include/linux/mroute.h). Before we start looking at the code, it is important to mention two other files: /proc/net/ip_mr_vif and /proc/net/ip_mr_cache. As we will see later, these two files are particularly useful for understanding the current state of the multicast router. The ip_mr_vif file lists all the virtual interfaces involved in multicast operations, while ip_mr_cache exactly represents the state of the MFC.

Now, as an example, we start sending out multicast traffic from VAIO to the address 224.225.0.1. Packets are received by SNOOPY but are not forwarded to eth1 because no JOIN has been issued by MAFALDA. When MAFALDA does issue a JOIN to 224.225.0.1, the contents of the two proc files (on SNOOPY) are shown in Table 1 and Table 2.

Table 1. Content of /proc/net/ip_mr/vif

Table 2. Content of /proc/net/ip_mr_cache

In ip_mr_vif the two interfaces (eth0 and eth1) involved in our multicast routing are listed, while the third interface (pimreg) is a virtual device registered by the multicast management code. In Table 2, you can see the route PIMd has built. Here, packets coming from eth0 (Iif: 0) with a source address equal to 192.168.1.118 (Origin: 7601A8C0) and directed to 224.225.0.1 (Group: 0100E1E0) are forwarded to eth1 (Oifs: 1:--ignore the second number at the moment). Now, we issue a LEAVE message from MAFALDA. After a couple of seconds the content of /proc/net/ip_mr_cache is updated, and the result is shown in Table 3.

Table 3. Content of /proc/net/ip_mr_cache after a LEAVE Message

We still have the same origin and group, but now the input interface has been changed to -1, and we have no output interfaces. This is because packets still

are coming in from eth0, but because nobody wishes to receive them on other interfaces, they are dropped. When packets are dropped this way, a new entry is stored in a special queue of unresolved addresses. To indicate this is an unresolved address, the input interface is set to -1 and no output interfaces are listed.

This operation of queuing unresolved packets is necessary for a particular reason: receiving an IGMP packet and adding the corresponding MFC entry takes a lot of time (about 2-3 seconds on our test network and about 20-30 seconds on a bigger network with interconnected multicast routers). When VAIO started transmitting multicast packets in our previous example, there was a high probability that the JOIN message from MAFALDA wasn't yet handled by SNOOPY, and the corresponding MFC entry necessary to forwarding packets was not already set up. So, in order not to lose the packets received from VAIO while the JOIN request is handled and the new MFC entry is added, they are queued in this special cache. As soon as the MFC entry is added, the queue is cleaned up, and the packets waiting in it are forwarded to the right destination. Obviously, due to performance and memory restrictions, this queue cannot grow too large. This is solved by simply adding a timer function that periodically cleans up the cache of unresolved entries (ipmr_expire_process()).

Now, let's take a look at the data structures involved in this process (Listing 1).

Listing 1. The vif_device and mfc_cache structures used by the multicast routing code.

vif_device is a virtual device linked to a real network adapter. The dev field is a pointer to the net_device structure representing the real hardware interface. More interesting is the mfc_cache structure. Its fields are self-explanatory and reflect all the data shown in Tables 2 and 3.

The three main variables used in ipmr.c are as follows:

```
/* Devices              */
static struct vif_device vif_table[MAXVIFS];
/* Forwarding cache     */
static struct mfc_cache *mfc_cache_array[MFC_LINES];
/* Queue of unresolved entries */
static struct mfc_cache *mfc_unres_queue;
```

vif_table is simply an array of all the virtual devices created by PIMd; mfc_cache_array represents the MFC; and mfc_unres_queue is the list of unresolved entries described above. Prior to analyzing the multicast management code, it is worth spending a couple of words on the TTL array, a member of the mfc_cache structure. Each value of the array is linked directly to the vif_table. In fact, for each multicast address assigned to every interface, we have a single byte value identifying the TTL threshold. As we will see later, this

value is compared to the TTL field of each IP packet when deciding if the packet is to be forwarded.

We have seen the basic data structure of multicast routing, so now let's take a look at how they are manipulated by the kernel. All the functions are implemented in one single file, ipmr.c. Keep in mind that the code in this file does not implement the routing protocol itself. The functions you can find in ipmr.c are used by the multicast routing dæmon (PIMd in our case) to manage these data structures. Simply put, whenever PIMd decides it's time to add or delete a route, it merely sends a message to the kernel specifying the action that should be taken. PIMd, in order to do that, must be able to receive IGMP packets; these are passed up to user space by the kernel. PIMd communicates to the kernel in two different ways: using ioctls and via the setsockopt() system call. Both the vif and mfc tables are handled using the setsockopt() system call.

In order to better understand how this is achieved we take a look at some of PIMd's code as well. In particular, all the functions communicating with the kernel are located in the kern.c file of the PIMd distribution. Here, the function k_chg_mfc() is responsible for adding or modifying an existing MFC entry, while the deletion of an existing entry is performed by k_del_mfc(). In order to tell the kernel how multicast packets should be forwarded, some information similar to that listed in the mfc_cache structures must be provided by the user dæmon. In particular, this information is encapsulated in a new structure defined as mfcctl (Listing 2).

Listing 2. The mfcctl used by PIMd.

The fields in this example should be self-explanatory. It's important to mention, though, the role of the mfc_ttls array in this structure. As stated earlier, this value represents the TTL threshold; however, it is treated in a slightly different way by the user-mode dæmon. The function k_chg_mfc() must specify to the kernel on which interfaces the multicast packet should be forwarded. In order to do so, a list of the output interfaces must be provided; the mfcc_ttls fills this role. The code snippet below shows this point:

```
for (vifi = 0, v = uvifs;
     vifi < numvifs; vifi++, v++)
{
if (VIFM_ISSET(vifi, oifs))
            mc.mfcc_ttls[vifi] = v->uv_threshold;
        else
            smc.mfcc_ttls[vifi] = 0;
}
```

Here, if an interface is indeed an output interface for a particular multicast address, its TTL threshold is set to the real value; otherwise it is set to zero. The kernel interprets the value zero as a non-output interface for that particular

group, and as a consequence, it will set the corresponding byte of the mfc_cache structure equal to 255 (decimal) and not forward the packets.

Now, let's see what the kernel does when it receives a request to add a new entry to the MFC. The type request is handled by the ipmr_mfc_add() function. The kernel checks whether this is an update request by looking for the current entry in the MFC. If a matching item is found, the new TTL's values are copied into the existing mfc_cache structure, and the minvif and maxvif values are updated as well. These values indicate the minimum and the maximum index values of all output interfaces for a particular multicast address. The function performing this job is ipmr_update_thresholds(). For your convenience, we include the function shown in Listing 3, because it better explains the meaning of the minval and maxval fields.

Listing 3. How the kernel updates a current MFC entry.

Back to our ipmr_mfc_add() function; we now consider the case where an existing MFC entry is not found. In this case, a new structure is allocated and inserted into the MFC table. Once this operation is completed, the kernel must perform one last action: forwarding any unresolved multicast packet currently queued in the mfc_unres_queue that might be directed to the newly added destination node. In an affirmative case, packets are removed from this queue and forwarded to the new interface. The other operation that remains to be executed is the MFC delete. This one is pretty straightforward—data structures are basically the same as what was seen before. In order to remove a cache entry, the user-mode dæmon invokes the k_del_mfc() function while the handler for the kernel mode invokes the impr_mfc_delete(). This function simply removes the specified entry from the MFC.

Now that we have identified where MFC entries are added, modified and deleted, we can start hooking the multicast routing code. Function hooking is a simple concept. Basically, a function inserted in the middle of the code shown previously causes the system to switch to an external function implemented in a kernel module. In a way, the function implemented in the module can be seen as a callback that is invoked every time an MFC entry is added, modified or deleted. To implement this hooking mechanism, we based our code on a well-known hooking architecture already implemented in the 2.4.x kernel: the Netfilter interface. If you have never used it before, suffice to say it's a popular interface implemented within the kernel to allow packet filtering. The same action that Netfilter performs with packets can be done easily with arbitrary data structures—mfc_caches in our case. In particular, the following is the prototype of the callback function:

```
typedef void nf_nfy_msg(
    unsigned int hook,
```

```
      unsigned int msgno,
      const struct net_device
      *dev,
      void* moreData);
```

Here, hook represents the domain (this value always will assume the PF_INET value); msgno represents the message number (the action taken by the kernel, adding an mfc_cache entry, for example); dev can assume the value of any current net_device involved during the operation; and moreData is a void* pointer to a generic data structure. This pointer is indeed a pointer to an mfc_cache data structure.

Now that we have seen the format of the callback let's see how it can be invoked by the kernel. It's actually quite simple; placing the following in the kernel code will invoke any registered hook function for the specified action:

```
  #ifdef NF_EVT_HOOK && NF_MCAST_HOOK
      NF_NFY_HOOK(
          PF_INET,
          NF_NFY_IP_MCAST_MSG,
          IPMR_MFC_ADD,
          NULL,
          (void*) c);
  #endif
```

However, the details about our hook architecture are beyond the scope of this article. The reader will understand the code better by looking at our modified kernel files [available from the *Linux Journal* FTP site at ftp.linuxjournal.com/pub/lj/listings/issue103/6070.tgz] or at the original Netfilter implementation.

Now that we have had a complete overview of the multicast routing implementation in Linux and a couple of notes about how to implement a hooking mechanism, it's necessary to pinpoint a couple of concepts we observed during our tests.

Let's go back to our testing network and imagine the following scenario: SNOOPY is sending out multicast traffic, and both MAFALDA and VAIO issue a JOIN request. You expect to see a new multicast route with eth0 and eth1 as output interfaces. Unfortunately, that is not what happens. Taking a look at /proc/net/ip_mr_cache, you can see only a single route to MAFALDA, but both MAFALDA and VAIO are receiving multicast traffic correctly. Here's why: outgoing packets from SNOOPY are sent using 192.168.1.200 as a source address. For that reason, when sending out data on eth0, SNOOPY will behave like it's sending out multicast traffic on a LAN. That means SNOOPY will start sending out packets on eth0 even before VAIO issues a JOIN, because the kernel is unaware of the presence of a PIM dæmon able to interpret IGMP packets coming from other PCs. In order to make other workstations receive multicast traffic, it simply sends out packets. This way, any other machine on the same network segment can turn on the multicast hardware filter and pick

up the desired data from the wire. In a similar situation, multicast JOIN and LEAVE requests cannot be hooked on the primary interface, because on that interface the kernel is not exactly performing multicast routing.

Matteo Pelati (matteo@dolce.it), while completing his studies, works as a research assistant at the University of Milan in Italy. His primary interests include network protocols, operating systems and traveling.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

OpenACS Packages

Reuven M. Lerner

Issue #103, November 2002

Using the APM application to install, distribute and remove packages specifically used in database applications.

One of the core ideas of software engineering is to divide a large project into separate modules. Modularization makes it easier to customize a system for your own specific needs, allowing you to write new modules and remove unnecessary ones. Using modules also makes it easier to distribute the work among many different programmers. A quick review of the available Linux, Apache, Perl and Python modules freely available on the Internet makes this point very clear.

OpenACS 4 (Open Architecture Community System), the toolkit for creating on-line communities that was initially examined here last month, dramatically improves on earlier versions in a number of ways. But perhaps the most important change is the division of functionality into modules, which are called “packages” in the OpenACS world. Because each package is self-contained, and because it is possible to connect any package with any URL, OpenACS 4 has made it easier than ever to create flexible community web sites.

This month, we take an initial look at OpenACS packages, including how we can install and use them. (This article assumes that you already have installed PostgreSQL, AOLserver and the core OpenACS functionality, as described in the last two installments of At the Forge.) Since most OpenACS sites use some of the functionality that comes with the built-in applications, rather than write everything from scratch, installing packages is something every OpenACS administrator needs to know how to do soon after installing the core system.

Web/Database Packages

Consider the following simple CGI program written in Perl:

```
#!/usr/bin/perl
use strict;
use warnings;
use CGI;
my $query = new CGI;
print $query->header();
print $query->start_html(-title => "Testing");
print "<p>This is some text</p>\n";
print $query->end_html();
```

If I install this program as test.pl in my web server's CGI directory, others can see the results of its execution by retrieving www.lerner.co.il/cgi-bin/test.pl. If I want this program to be available under a number of different names, I can copy it; the name that I choose will be reflected in the URL.

Things get a bit trickier if my server-side application consists of several CGI programs rather than a single program. If I want to have several copies of such an application suite running on my system, I must copy all of the program files. In many cases, it'll be easier to place all of the files in a directory, then copy the directory and all of its contents each time I want the application to run somewhere else.

Making such copies carries potential synchronization problems: if I fix a problem in one copy of a program, I will have to make the same change to every copy of the program. I can resolve some of these problems with CVS, but I also could eliminate this issue by keeping only one copy of my program on the filesystem. Then I could configure the web server (either Apache or AOLserver) to treat one or more URLs as requests for my program.

Now consider what happens if this application suite takes advantage of a relational database. Installing the application is no longer as simple as copying files or configuring the HTTP server. Now, we also need to have some way of resolving potential conflicts and confusion between the copies of a single application, such that the forums at /foo/bboard don't get confused with /bar/bboard in the database. If and when we remove our application from the system, we also will need a way to remove the database tables it used.

In OpenACS, the solution to this problem is APM, the ArsDigita Package Manager. APM was originally written by ArsDigita, a now-defunct consulting company that wrote the predecessor to OpenACS. ACS worked only with an Oracle database server, whereas OpenACS works with both Oracle and PostgreSQL.

APM handles a number of different issues inherent in server-side applications that use a database, including version control, scripts for table creation and removal and database independence. APM also has been designed to allow each copy of an application to have independent configuration variables and to be associated with one or more separate URLs.

Filesystem Layout

An APM really is nothing more than a .tar.gz file with an .apm extension. The file is typically named like this: *packagename*-0.5d.apm—where *packagename* is the unique name associated with the package. This example package contains development version 0.5. Opening a package with **tar -zxvf** reveals a standard file and directory structure:

- packagename.info, an XML file describing the contents of the package. This file, normally created automatically by the OpenACS APM application, tells OpenACS which files are associated with the package and which configuration parameters are available for the user. It also indicates whether the application is a singleton (i.e., provides services for the rest of the system) or an application (i.e., can be run from a particular URL).
- The sql directory is where the table-creation (and table-destruction) scripts are located. Originally, when ACS supported only Oracle, this directory normally would contain two files: *packagename*-create.sql and *packagename*-drop.sql. The APM installer would run the **create** script when the package was installed and the **drop** script when it was removed. (The create script often runs INSERTs as well, seeding database tables with standard data for later use.)

Now that OpenACS supports PostgreSQL as well as Oracle, this directory structure has changed somewhat. Within the sql directory are oracle and postgresql directories that have parallel scripts for creating and dropping the tables. Each installed copy of OpenACS knows which databases it supports (based on the value of a variable in AOLserver's nsd.tcl configuration file), and thus chooses the most appropriate script.

- The tcl directory contains Tcl files containing procedure definitions. These procedures are loaded into AOLserver at startup time, giving them a speed advantage over those defined inside of .tcl (or .adp) pages elsewhere in the OpenACS system.
- The www directory contains what we normally expect to be associated with a web application. This is where we put our .tcl and .adp pages, as well as any graphics and auxiliary files associated with the application. OpenACS's query dispatcher, which makes it possible for server-side programs to support multiple database servers, works with XML files with an .xql extension; these also go in the www directory.
- Because of how the OpenACS templating system works, it's not unusual for a single web page to use three files: a .tcl file for setting variables, an .xql file that defines the SQL query used to retrieve rows from the database and an .adp file that is responsible for turning the information into HTML.

APMs also may contain a number of other files, such as database upgrade and migration scripts (for those users who are upgrading from a previous version of the package), regression tests (to ensure that the package works correctly), administration facilities (under www/admin) and HTML-formatted package documentation (under www/doc).

Loading and Installing Packages

The first step in working with an APM package is to load it, which normally means copying it into the filesystem where your OpenACS system resides. If your OpenACS system is under /web/atf/, then all packages go under /web/atf/packages. (For this reason, each package needs a unique name; many OpenACS developers use an Emacs-style package naming convention, in which the package name is preceded by the developer or client name. This helps to avoid conflicts between the foo-attributes and the bar-attributes packages.) Copy the package's entire directory structure into /web/atf/packages, making sure the files and directories are readable (and writable) by the user ID under which AOLserver operates.

An easier and more reliable way to install APMs is to click on the load packages link within the package manager. OpenACS will ask you for the URL of the APM or the name of the directory in which one or more packages reside. OpenACS then will find all of the .apm files located there, unpack and load them into the system.

Once a package has been loaded into the filesystem, you must install its data model and register it with the system. This is done through the web-based package manager, which normally is found under the URL /acs-admin/apm on an OpenACS system. Typically, only the site administrator has access to the package manager.

The main package manager screen shows all of the packages that are loaded in the system and indicates whether each has been installed, superseded by a newer version or is yet to be installed. Using the menu options at the top of the page, you can ask to see different subsets of the packages on the system, including only those that you are personally responsible for developing and managing.

The package manager is the main way in which you create, modify, update and install packages:

- You can install the data model for a package and register the package with OpenACS. The next time OpenACS restarts, files in the package's tcl subdirectory will be loaded into AOLserver's memory. Your package can be connected to a URL via the OpenACS site map, as we will soon see.

- You also can use the package manager to create a new OpenACS package. Indeed, the first step in creating a new OpenACS package is to use the package manager to set up a new directory and .info file.
- You can examine any package currently loaded into the system and retrieve a list of parameters, files or any other information associated with the package.
- You can modify a package, changing parameters, files and other information associated with the package.

To install a new package, click on the install packages link at the bottom of the page. The package manager will scan the packages directory for any new packages, allowing you to choose which packages should be installed. (When you first install OpenACS, no packages are installed, so this is a very long list.) Each package can depend on one or more other packages. If you try to install a package without its prerequisite dependencies already installed, the package manager will require confirmation before continuing.

The installer lets you decide whether you want to install a package's data model or also enable the package for use on the site. Personally, I always enable any package I install, but I'm sure there are reasons why you might not want to do this. After checking the appropriate boxes, click on the Next button, which installs the data model. Following this, you must restart the AOLserver, because many packages depend on Tcl libraries loaded into AOLserver at startup. Therefore, the packages will not work until the server has been restarted.

Mounting a Package

Once you have restarted AOLserver, go to the package manager and look at the list of enabled packages. All of the packages you loaded should be visible on this list. From here, you can modify the packages, load more packages or begin to turn the loaded packages into an actual web site. This process is known as mounting and is performed using the OpenACS site map. In one of the more confusing parts of OpenACS administration, the site map is not part of the site-wide administration page; rather, it is part of the main site administration page. In other words, you manage packages under /acs-admin/apm, but you manage the site map under /admin/site-map. There are some good reasons for this, but it tends to confuse people more than it helps them.

The site map tells OpenACS how to connect a URL to an application. For example, you might want the OpenACS bboard package to be under the /forum URL or the /bboard URL. In some cases, you actually might want to have it in both places. The site map allows you to do this by clicking the mouse.

To connect a package to a URL for the first time, click on the New subfolder link to the right of the / path. You are asked to name the URL under which the new application should be mounted. To install the bboard package under /forum, you would enter **forum** (without the leading slash).

If you stop here, you can treat the new subfolder as nothing more than a folder in which new folders and/or static documents are placed. But you also can click the New application link associated with this folder, choosing an installed application package and giving it a human-readable name that will be used in titlebars and headlines. So while you might put the forums under the /bboard URL, you might want to give it the name Discussion forums. This title cannot be changed all that easily, so give this process some thought.

The new application link creates a new package instance and then attaches it to the directory you have created. To make an alias to this application, you can create a new subfolder and then use the mount link. It took me a while to understand that mount lets you connect to existing application instances, while new application creates an entirely new instance. This makes more sense when you consider that unmounting an application (using the unmount link to the right of the pathname) does not delete it but makes it unreachable. To delete an instance of an application completely, you must click on the unmounted application link from the site map, and then click on the delete link next to the unmounted application.

Each instance of an application has its own permissions and its own parameters. I have found parameters to be a particularly useful part of OpenACS, in that they let me create an application once and use it many times, but give each instance its own configuration. Following the appropriate links on the site map, you can view or change the parameters associated with a package instance.

Conclusion

OpenACS packages, distributed as .apm files and managed with the APM application within CVS, make it possible to create and distribute web/database applications. Once imported, an APM package can be instantiated multiple times, each with its own permissions and associated parameters. As we will see next month, you also can use APM to create your own web/database application packages and distribute them easily to coworkers and other community members.

Resources

email: reuven@lerner.co.il

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

Archive Index Issue Table of Contents

Advanced search

Serving Up the All-Linux Office

Marcel Gagné

Issue #103, November 2002

Some simple and cost-conscious ways to migrate workstations to the Linux desktop.

Ah, François, the restaurant looks wonderful tonight. Do have another look around, and make sure that each workstation has the necessary tools. I want this to be perfect.

Bonsoir, mes amis, and welcome to *Chez Marcel*, home of fine Linux cooking and marvelous wine. Please sit down and be comfortable. François, run along and fetch the wine. *Vite*! Better make it the 1999 Châteauneuf-du-Pape.

Those of you who have been visiting my restaurant for some time no doubt will take pleasure in the fact that many companies (and individuals) are looking for alternatives to their desktop operating system. Linux is the natural choice and offers many relatively painless ways to make that switch. Unfortunately, modern Linux distributions running KDE and GNOME can demand a great deal of system resources, and many companies looking to make a switch have modest PC hardware. Luckily, Linux makes it possible to set up workstations with minimal Linux installations that offload most of the work to a powerful, central server.

This type of setup is the thin-client design. Exactly how thin this client can be depends on which approach you choose. As you would expect with all fine ingredients, there are many ways to prepare something. Fresh lobster steamed and served with drawn butter and garlic is certainly excellent, but it is not the only way. The same holds true for what we prepare in our Linux kitchens, in this case thin-client implementations. In fact, one approach lets you set up Linux workstations without re-installing the client PC.

Probably the easiest way to achieve our thin-client goal is to run a Linux workstation with an X server and not much else. Rather than starting up KDE or GNOME, you can use OpenSSH:

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

The resulting desktop is far from interesting. You get a plain old X desktop with nothing but an X terminal running and no window control whatsoever. What can you possibly do with this? For one thing, you can connect to your server using OpenSSH, like this:

```
ssh -C -X -l user_name server_name
```

The -C option sets up data compression in the secure stream, while the -X enables the forwarding of X requests. Once you find yourself at the command prompt, you can launch a desktop session on the remote host by typing **startkde** or **wmaker** or whichever desktop you prefer. Suddenly, you will find yourself running a full graphical desktop session on the remote server.

Note that the server's /etc/ssh/sshd_config file must have X11 forwarding turned on before the client request can be accepted. Here is what the parameter in that file should look like:

```
X11Forwarding yes
```

If you needed to change the parameter, you also will need to restart sshd with **service sshd restart**. This kind of a remote desktop setup works wonderfully and can be quite useful even if you already run a graphical Linux desktop. It is particularly useful if having access to a desktop on the server would otherwise require you to go to the computer room, which happens to be across the building.

If you want to run your own desktop *and* one on the server, switch to one of your virtual consoles by pressing Ctrl-Alt-F2 (or F3 or F4...) and start a second X session like this:

```
startx /usr/X11R6/bin/xterm -- :1
```

The -- :1 option (note the double hyphen) brings up a second graphical desktop —yet another boring xterm on a boring background. From here, you can use **ssh** to connect to your server and start whatever desktop session you like. To switch back to your regular session, press Ctrl-Alt-F7 or Ctrl-Alt-F8 for the second session. The downsides of this setup are that this isn't exactly *thin* and, even with compression, the OpenSSH stream can seem a bit slow.

The best way to do an X display session (and to thin down the client) is to use XDMCP, the X Display Manager Control Protocol. From the Linux command line on the client PC, you would simply type:

```
X -query server_name
```

Magically, you would have a graphical login screen exactly as it is running on your server. Ah, *mes amis*, as you might expect, a little preparation must go into this recipe before we can execute that simple command, but the preparation is simple. While François refills your glasses, I will explain.

In order to get a graphical login screen presented to a querying host, the /etc/X11/xdm/Xaccess file needs to have the following lines uncommented:

```
*                       #any host can get
                            #a login window
*     CHOOSER BROADCAST  #any indirect host
                        #can get a chooser
```

You also need to have xdm running (or kdm or gdm), which means your server comes up in runlevel 5. Those of you who boot up into a graphical login screen probably can skim the next few lines. To start services in runlevel 5, you must set the initdefault in your /etc/inittab file to 5:

```
id:5:initdefault:
```

If you aren't booting up into a graphical desktop, the 5 above should be a 3. One last thing needs to be set in that /etc/inittab file. Look down at the bottom, and you should see something like this:

```
# Run xdm in runlevel 5
# xdm is now a separate service
x:5:respawn:/etc/X11/prefdm -nodaemon
```

Log out and restart your X server so that the new changes take effect. Then, from the workstation, simply run **X -query *ip_address*** (the IP address of your server, of course). If you are running XDM, things should work. But, many of you will be using KDE's KDM login manager, and you may find yourself looking at a flat gray screen with an X in the center (although you can move that X around, it isn't very interesting). That's because XDMCP access to KDM requires a little bit more tweaking.

KDE uses a kdmrc file. On my system, it is located in /etc/kde/kdm. Look for the following lines and make sure that the Enable parameter in the [Xdmcp] section is set to true:

```
[Xdmcp]
# Whether KDM should listen to XDMCP requests.
Enable=true
```

As François brings out the double-butter Brie to go with your wine, let me tell you about the thinnest clients of all, diskless workstations. Several options are available for this approach; I will concentrate on one for today. However, given how important a tool this is for Linux desktop deployment in the workplace, I feel it is my duty as proprietor, in this home of fine Linux fare, to point out a handful of other projects worthy of your consideration. I will share some of these with you before closing time, but for now, let's talk about booting these workstations.

Boot diskettes, with the appropriate driver for your Ethernet card, can be created using the tools from the Etherboot web site, etherboot.sourceforge.net. Building from source is simply a matter of untarring the distribution, changing to the src and typing **make**. There is an even easier way to do this, however, thanks to the folks at ROM-o-matic.net (rom-o-matic.net). They maintain a collection of boot ROMs generated from the Etherboot Project. Simply select the release level (latest is usually best), the card type (3c509, tulip, etc.) and click Get ROM. The downloaded file then can be transferred to a diskette:

```
cat eb-5.0.6-yournic.lzdsk > /dev/fd0
```

Et voilà! Our diskette is done. Pop it into the workstation PC and reboot. When the system comes up off the card, it will pause for a few seconds by default, giving you the opportunity for a local or network boot—network is the default. The problem is, your client has nothing to connect to after it boots. This is where the Linux Terminal Server Project at www.ltsp.org comes into play. The whole point of the project is to make the setup and operation of thin-client Linux as simple as possible, *non*?

All right, let's put that workstation out of our minds for a few minutes and concentrate on the server. You will need a couple of additional things, starting with a DHCP server (the dhcp package from your Linux distribution) and a TFTP server (most likely, the tftp-server in your distribution). You'll also need to install the packages for your NFS server. Any major distribution will have these on the CD(s), so you won't need to look far for them.

In the case of TFTP, you will find yourself with an entry in /etc/xinetd.d for the service. Unfortunately, although the package install will create an entry, it also will be disabled by default. So, make sure the service (/etc/xinetd.d/tftp) has this line:

```
disable = no
```

Then, restart xinetd to activate the TFTP dæmon.

If you haven't already done so, now would be a good time to get the LTSP packages themselves. The site does offer tarred and gzipped packages for everything, but it also provides precompiled RPM packages for Red Hat, SuSE, Mandrake and others. Debian users also will find prepackaged DEBs available. If you choose to work from the tarred bundles, simply extract them and run the install.sh script in each package. Speaking of packages, these are the ones you will need—I'm listing RPMs here, but the packages are in the order you need them: ltsp_core, ltsp_kernel, ltsp_x_core and ltsp_x_fonts. There are a few additional local application packages, such as Netscape, but we will leave them for now.

This is the spot where things got a bit interesting on my setup. I installed LTSP on three different machines and found that their order was quite important. I needed to have dhcpd running for the LTSP configuration step, but starting the dæmon required that I have a valid /etc/dhcpd.conf file in place. Because I didn't have such a setup, the autoconfig was failing so I decided to create my own dhcpd.conf file; it isn't difficult. Have a look at the following configuration. It should be sufficient to get you going, keeping in mind, of course, that the IP address of my domain (and my root-path server IP) certainly will be different from yours:

```
subnet 192.168.22.0 netmask 255.255.255.0 {
    option domain-name "yourdomain.dom";
    option root-path "192.168.22.10:/opt/ltsp/i386";
    range dynamic-bootp 192.168.22.128 192.168.22.254;
    default-lease-time 21600;
    max-lease-time 43200;
    }
host baroque {
    filename "/lts/vmlinuz-2.4.18-ltsp-1";
    hardware ethernet 00:50:FX:5B:XX:56;
    fixed-address 192.168.22.5;
            option host-name "baroque";
    }
```

Note that I have created at least one host. The filename is correct for the latest LTSP distribution, but the hardware or MAC address for the Ethernet card (besides being fictional here) must be a valid one for your client. Start dhcpd with the command **service dhcpd start**.

Once you've installed your DEBs, RPMs or tarred packages and started the various services, you can thank the folks at LTSP because the other work pretty much has been done for you. All you need to do is execute an initialization script, which is provided:

```
cd /opt/ltsp/templates
./ltsp_initialize
```

After you type the command, you will be warned that the script is about to update some important system files, specifically /etc/dhcpd.conf, /etc/exports and a few others (Figure 1).

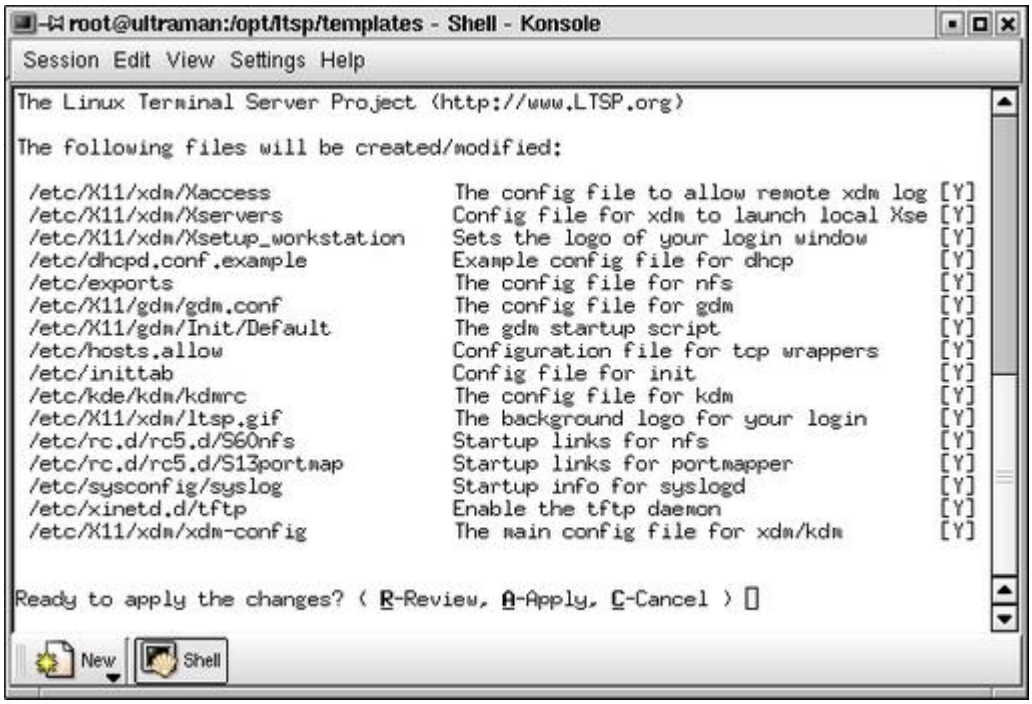


Figure 1. Configuring Your LTSP Distribution

Since NFS has been updated with new exports and DHCP configuration information has been changed, you should restart the NFS dæmon as well as the DHCP dæmon:

```
service nfs restart
service dhcpd restart
```

We are almost done, *mes amis*. In seconds, we will have a top-of-the-line, yet modest, Linux thin-client installation. Because all the software for LTSP is in the /opt/ltsp, we need to make a little side trip there for some configuration. Start by editing the file /opt/ltsp/i386/etc/lts.conf. Specifically, look for this line:

```
SERVER = 192.168.22.10
```

Make sure that it reflects the IP address of the host you are using as your server. This lts.conf file is a good one to get to know. It allows you to tweak individual workstation configurations on the off chance that the defaults don't work for a particular one. Here's a configuration I created for my workstation called baroque (it needed to be configured with an old serial mouse):

```
[baroque]
        XSERVER             = vesa
        X_MOUSE_PROTOCOL    = "Microsoft"
        X_MOUSE_DEVICE      = "/dev/ttyS0"
        X_MOUSE_BUTTONS     = 3
        LOCAL_APPS          = N
        SWAPFILE_SIZE       = 64m
        RUNLEVEL            = 5
```

Remember, as you add workstations (and bring more on-line), you need to put these in the /etc/dhcpd.conf file. Yes, you do need to restart the dhcpd service when you make changes, but that is all.

Now that we have that out of the way, put in your diskette, boot up your workstation, and in a few seconds, you should see your kernel image being downloaded to the workstation via TFTP. A configuration boot-time dialog will scroll past, followed by an automatic X configuration. Incidentally, if X is giving you a hard time at this stage of the game, you can change the workstation's entry in the lts.conf file to runlevel 3 instead. You'll then boot up into a text-only screen.

Closing time is fast approaching, *mes amis*, but I did say I would tell you about similar projects. You may want to look at Solucorp's X terminal toolkit (www.solucorp.qc.ca), headed up by Jacques Gélinas of Linuxconf fame. You also might pay a visit to Diego Torres Milano's site at pxes.sourceforge.net and have a look at his PXES Linux thin-client package.

Thank you, *mes amis*, for sharing this evening here at *Chez Marcel*. I do hope that your experience in thinner Linux has been a good one and that you will consider it in your place of business. The cost benefits along with simplified administration make it a perfect choice in this cost-conscious world. Update the server, and you update everyone; back up the server, and you've backed up every workstation—it doesn't get any easier than that. François, please refill our guests' wineglasses a final time. Until next month. *A votre santé*! *Bon appétit*!

Resources

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

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Hey USA, Don't Miss the Boat!

David A. Bandel

Issue #103, November 2002

Thanks to open minds, Linux is catching on in Panama.

When many folks think of emerging markets for Linux, I'm sure they look at the success stories of DreamWorks and many others. While these may be good markets for Linux, they're not the only ones. In fact, the real emerging markets are outside US borders in third-world countries, such as Panama, where I'm presently living. When I arrived in Panama, almost no one I had contact with had even heard of, much less tried, Linux. That's changing rapidly. It started slow, but is growing faster every day; a few firewalls here, then a few mail servers. Now, especially with the help of some specialized distros like Knoppix (www.knopper.de), which allows people to test-run Linux without buying a new drive or wiping out their old OS, Linux is taking over the point of sale and desktop strongholds. It's going slower in the US, where the entrenched mindset is hard to break through, but here, the minds are as open as the wallets are empty.

My four- and seven-year-old daughters both log in to Linux to surf the Net and play games, and they have no clue any other OS exists. The Barbie.com and CartoonNetwork.com/PowerPuff Girls web sites are among their favorites. And *Galaga* and *Gcompris* keep them entertained off-line. Don't let America loose the edge; your kids can learn to love Linux too. So when thinking of emerging markets, think young and global.

Genmenu projects.gtk.mine.nu/genmenu

This Bash script will generate a menu for Blackbox, Fluxbox, Enlightenment or Window Maker. You don't need to be root to use it, because the edits will be written to your $HOME directory. I do suggest you make a backup first, as Genmenu will clobber your old file at the moment (though that may be fixed by the time you read this). If you know Bash scripting, you can easily change this script to look for your favorites (or not look for some others). Using it could

help maintain a large number of systems uniformly, if it's run during each login. Requires: Bash.

HTun htun.runslinux.net

If you need to bypass a firewall that has only one or two open ports, but you need to connect to other systems on different ports, HTun may be your answer. You will need to run HTun on an internet-connected system (your home system perhaps), but you can create a VPN and use your home system as a proxy to reach those systems not accessible from behind your firewall. You'll want to take some security measures to make sure your internet-connected system isn't abused, but this is a handy tool for working around restrictive firewall policies. Requires: libpthread, glibc.

Shared Recording Server oktober.stc.cx/source/erec.html

erec is designed for raw audio streams, and it works by attaching to a sound device and acting as a front end. As it listens, it opens port 6130 so other programs (even multiple programs) can attach and listen to whatever sounds are coming from your sound card. Requires: libargh (included), libstdc++, libm, glibc.

Simple Expense Manager www.angelfire.com/tn3/petbath

This expense manager is so easy to use, you might use it solely for that reason. Until now I added my expenses into my accounting program, but sometimes that's too hard to do every day. While these transactions still need to be added to my accounting program, I can do it as aggregated totals at the end of the week or month instead of fighting with it daily. Requires: Perl, Perl module Apache::Htpasswd.

ProBIND probind.sourceforge.net

I'm always looking for better ways to handle DNS administration (among other things). This particular application does a good job, although it is lacking in the security area if you have multiple users updating only their DNS server. It works extremely well if either users are all trusted or only one person is updating many zones. Another drawback is the requirement to have the PHP CGI module, the standalone interpreter. But if you have a lot of DNS zones to handle, I'd look closely at this application. Requires: MySQL, PHP with MySQL (both as a module and as a standalone interpreter), Apache, Perl, Perl module Net::DNS, OpenSSH (optional).

graphical Process Statistics www.gps.seul.org

Among the choices of software I looked at three years ago were DHCPXD, still a great choice for a DHCP client, and gPS. A lot of good software exists that will show you what you need to know about processes running on your system; the ps and pstree CLI utilities come to mind. But if you want to see something different using CLI, you have to rerun it, and gPS will change dynamically. You even can watch processes on other systems by running an included remote gPS poller on the remote systems. Requires: libgtk, libgdk, libgmodule, libglib, libdl, libXi, libXext, libX11, libpthread, libstdc+ +, libm, glibc. Until next month.

David A. Bandel (david@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is coauthor of Que Special Edition: Using Caldera OpenLinux.

Archive Index Issue Table of Contents

Advanced search

Why We Still Oppose UCITA

Lawrence Rosen

Issue #103, November 2002

Vagueness, inapplicability and measures that fall short—why we need to start fresh.

An attorney for Red Hat recently asked me to join her in requesting that the National Commissioners on Uniform State Laws (NCCUSL) reverse their 1999 decision to adopt UCITA, the Uniform Computer Information Transactions Act.

I've commented before on UCITA [see *LJ*, June 2002]. Readers of this column will recall that UCITA is a model code intended to be adopted by all states, so there is uniformity within software licensing law. UCITA provides default rules that apply when a software license omits essential terms. Another purpose of UCITA is to define what license terms are against public policy and thus cannot be enforced, even if they are included in a license.

On behalf of the Open Source Initiative, I wrote to the NCCUSL to oppose UCITA. I did so because UCITA does not address yet many of the major concerns of licensors and licensees of open-source software. Even though recent amendments to UCITA have begun to recognize our unique issues, the proposed law remains flawed, incomplete, confusing and biased toward licensors of proprietary software.

The drafters of UCITA have proposed several amendments to address our issues, but they still struck out with us. Here's what they proposed and why we continue to oppose it.

One recent amendment provides that “a copyright notice merely giving permission to use the software that is not part of a contract is not within UCITA.” This either is a truism (federal copyright law preempts state contract law anyway) or is inapplicable to the many open-source licenses intended to be contracts. I understand that this amendment purports to address the concerns of people using the GNU General Public License (GPL), a license whose author

urges that it be treated exclusively as a copyright license. What about all the other licenses that satisfy the Open Source Definition (www.opensource.org/docs/definition.php) and whose authors intend to form a contract? This provision is of no help; it simply does not matter.

Another amendment excuses licensors from implied warranty obligations "if the software is free (no intent for profit or commercial gain from the transfer of the copy or from controlling use or distribution of the copy)." This amendment relies on a *commercial* definition of "free" as "free of charge" rather than the far more important conveyance of rights to use, copy, modify and distribute software, along with access to the source code that makes those rights meaningful.

The latter concept of freedom underlies the principles of the Free Software Foundation (www.fsf.org) and the Open Source Initiative (www.opensource.org), but it apparently plays no role in UCITA. The actual language in the UCITA provision is vague and confusing, relying as it does on phrases like "intends to make a profit" and "acts generally for commercial gain". It will allow proprietary software vendors who hide their source code and limit the rights to copy, modify and distribute software to obtain the benefit of warranty exemptions, even though they actively obstruct their customers' ability to make the software "merchantable" and "fit for a particular purpose" by doing so.

A third amendment says reverse engineering for the purpose of interoperability cannot be prohibited by a license. This is an important step—albeit a baby step—toward affirming the fair use rights so badly damaged by the passage of the Digital Millennium Copyright Act. Unfortunately, because of federal preemption this provision is probably of limited effect. Furthermore, this idea is not the same as a strong statement by NCCUSL that a license provision that restricts or limits *any* fair use rights to software is unconscionable and against public policy. Such a broad provision would not solve the preemption problem, but it would make a valuable statement that may encourage Congress to restore the public benefit objectives that underlie copyrights and patents in the US Constitution. I am afraid that the current weak and limited UCITA amendment relating to reverse engineering will lull people into thinking that their former rights have been restored.

I believe that it will be important to start afresh with UCITA and consider the new environment in which open-source software competes against proprietary, closed software marketed by wealthy companies. UCITA is not particularly helpful to guide courts in interpreting or enforcing open-source licenses or to guide Congress in restoring fair use rights to the public. Without that, the Open Source community doesn't need UCITA.

Legal advice must be provided in the course of an attorney-client relationship specifically with reference to all the facts of a particular situation and the law of your jurisdiction. Even though an attorney wrote this article, the information in this article must not be relied upon as a substitute for obtaining specific legal advice from a licensed attorney.

email: lrosen@rosenlaw.com

Lawrence Rosen is an attorney in private practice, with offices in Los Altos and Ukiah, California (www.rosenlaw.com). He is also executive director and general counsel for Open Source Initiative, which manages and promotes the Open Source Definition (www.opensource.org).

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #103, November 2002

Letters from our readers.

Linux Support or Hot Air?

I think it's about time for *LJ* to cover companies that claim to support Linux with their products, but fail to deliver. InterVideo (www.intervideo.com) claims to have ported their windvd player to Linux, but their web site keeps telling us this:

> LinDVD, InterVideo's Linux software DVD player, is currently available only to manufacturers for evaluation and integration. Linux users should be aware that we are engaged with top computer, Internet appliance, and set-top box manufacturers to provide the highest quality DVD playback for their devices.

Cyberlink claims to have ported PowerDVD. Status: same as above.

Bioware keeps telling paying Neverwinter Nights customers that the Linux client will come, but every week the reason why it doesn't come yet is a different one. Last week, the coders were on vacation; this week we can't get status updates because “the PR people will all be at Gen Con.”

—Mathias Homann

We're an Element in a What Net?

Congratulations on your 100th issue! *Linux Journal* has been for years one of the critically important elements in the value-net that surrounds the Linux development community and is one of the reasons why Linux has become the fastest-growing operating system for platforms ranging from wristwatches to

supercomputers. I'm looking forward to the next 100 issues. Keep up the good work!

—Daniel Frye, IBM Linux Technology Center

Our Calendar Artist Makes It Big!

Thank you for all the calendars! I am extremely impressed with what you have managed to put together from my renderings and proud to hand the calendars out to my friends. I have already been contacted by an American company who had found my web site via the calendar, and they offered me a gig!

—Robert Karlsson

Nice Box, but How Much?

Loved the article on the Ultimate Linux Box [*LJ*, September 2002]. Just one complaint—no prices anywhere. It would have been great to see what you paid, even if that was super-secret pricing.

—Eric

PC component prices in effect when we write something are usually insanely high by the time you get your LJ. Anybody got a model for predicting them?

—Editor

Don't Run Microsoft Ads!

I'm glad to see issue 100 out. *LJ* has been a great source of inspiration to many of us all along. You'll keep my subscription for as long as you don't replace the true Linux/Open Source spirit with advertising from Microsoft and, to add insult to injury, come up with some cynical “justification”.

—Renato Carrara

Linux Journal Flunks Math

You used 8 Maxtor drives in the “Ultimate Linux Box” [*LJ*, September 2002] to get 1TB of storage. Now, if these are 120GB drives, 8 of them adds up to almost 1TB, but not if you use RAID-5, because you need to use at least one disk for parity partition, so you are down to 7.

—Fedor

We used 160GB drives and should have said so in the article.

—Editor

. . . and English

"How a Poor Contract Sunk an Open-Source Deal" [*LJ*, August 2002] is simply wrong. It should be "Sank", of course.

—Jacob E. Goodman

No Mercy for Memory Leaks

There is another system for checking for memory overruns and leaks that was not mentioned in Cal Erickson's "Memory Leak Detection in Embedded Systems" [*LJ*, September 2002]. The bounds checking patches for GCC can check local and static variables in C modules, which makes it much more powerful than a malloc debug library. Check the GCC Extensions page at gcc.gnu.org/extensions.html.

—William Bader

Somebody Ask Her What Month It Is

Thanks for the Tux calendar, well done. I've been a subscriber for several years, and have often been surprised and pleased—I think this is the first time I can say delighted.

—Carla

Old-School Power Mailer

On page 64 of your September 2002 issue you gave the Editors' Choice Award for Communication Tool to the Evolution mailer, saying "We like the idea of being able to compose more than one message at once." It has been possible to do this for more than a decade using MH, and now nmh. This suite of e-mail commands fits nicely with the UNIX shell, providing separate commands for scanning a mail folder, searching e-mail and composing new messages. Draft messages sit in their own folder giving an effectively unlimited number. Myself, I still have drafts to finish dating back to 1999! For more information on nmh, see www.mhost.com/nmh or www.ics.uci.edu/~mh/book/.

—Ralph Corderoy

More Lycoris Coverage Please

Is there any reason why *LJ* never writes a word about one of the latest Linux distributions from right here, the Eastside? I am talking about Redmond Linux, now called Desktop Linux from Lycoris. What gives? They have not bought any advertising space?

—Joe Pannon, Bellevue, Washington

Debit on the Left, Credit on the Right

I would like to know what accounting program to suggest when I am working with local businesses switching to Linux. I am interested in seeing a review and comparison of accounting software such as Quasar and Appgen MyBooks.

—Robert Tetzlaff

Ultimate No More

I always read with interest your articles on the Ultimate Linux Box. Each year was more powerful than the last. But I had other things to take care of, so I would read and dream.

Last year was different though, I was actually able to build, albeit slowly, your dual Athlon screamer! Was I happy? No way to describe it. I finally finished it this past summer with the purchase of a couple of fast SCSI drives. What a difference that made! What more could I possibly want except the nice case with all that wonderful cooling. But that will have to wait.

Why wait you ask? It's because in this latest (September 2002) issue you have now written another of these articles, and now I no longer have what you call the Ultimate Linux Box! I can't believe it? Do I go with the upgrade thing again? What is a guy to do? I worked so hard to get up with the rest of the fast crowd, and now I won't be at the top anymore. What a letdown. I'm totally disillusioned. I just don't know what to do. Perhaps next year when I read the article I'll have a chance to upgrade again. Until then, keep up the great work you do for the Linux community.

—Mike

Archive Index Issue Table of Contents

Advanced search

UpFront

Various

Issue #103, November 2002

They Said It, *LJ* Index and more.

Mandrake Number One in Desktop Survey

DesktopLinux.com has a page where readers are polled about their choice of desktop Linux distributions. So far 3,466 votes have been tallied. The poll question was “Which Linux distribution(s) do you use (or plan to use) on your desktop computer system?” Here's how the answers sort out:

1. Mandrake: 29.3%

2. SuSE: 14%

3. Red Hat: 12%

4. Debian: 10.2%

5. Elx: 9.1%

6. Lycoris: 6.8%

7. Gentoo: 6.4%

8. Slackware: 4.1%

9. Lindows: 1.8%

10. Libranet: 1.7%

11. Peanut: 1.3%

12. Xandros: 1.1%

13. Other: 1.6%

—Doc Searls

LJ Index—November 2002

1. Multiple by which the new RealNetworks Helix Universal Server running on Linux exceeds the speed of Microsoft's streaming server: 4
2. Number of Linux Users' Groups (LUGs): 500
3. Number of countries with LUGs: 80
4. Millions of dollars in Linux sales in 2001: 80
5. Projected millions of dollars in Linux sales by 2006: 280
6. Billions of dollars in Microsoft Windows sales in 2001: 10
7. Linux annual growth rate percentage according to Tower Group: 30
8. Linux annual growth rate percentage according to IDC: 37
9. Linux percentage of corporate IT budgets: 9
10. Percentage of retail sector CIOs “looking at open-source software”: 32
11. Number of IBM telecom customers who use Linux: 50
12. Number of IBM ISVs with Linux-enabled apps: 3,800
13. Number of Linux support personnel at IBM: 5,000
14. Thousands of Zaurus PDAs Sharp reportedly brought to LinuxWorld Expo in August 2002 expecting all would sell: 3
15. Number of New Zealand's Compaq servers replaced by one IBM Z Series mainframe: 150
16. Minimum number of IBM telecom customers using Linux: 50
17. Number of Linux instances on a Z Series mainframe at Solomon Smith Barney in New York: 62
18. Number of Linux instances on Korean Airlines IBM Z series mainframe: 10
19. Number of Korean Airlines personnel using the flight schedule system now consolidated on the mainframe: 4,000

Sources

1. 1: RealNetworks
2. 2, 3: Linux Counter
3. 4-6: quoted in CNET
4. 7-9: cited by IBM
5. 10: Tower Group, cited by Open Forum Europe
6. 11-13, 15-18: IBM
7. 14: Source at LinuxWorld Expo

“Radio” over IP over Radio

Here's another good reason why the Sharp Zaurus deserved the Editors' Choice we gave it two months ago. It's a WiFi radio: a wireless internet radio receiver. The first of the breed, in fact.

All you need is your Zaurus, a WiFi (802.11b) wireless card and a $10 shareware application called Zradio. Add headphones, and you've got a WiFi Walkman. Add a pair of portable powered speakers, and you've got your very own Linux boom box.

For more information, visit myZaurus.com.

—Doc Searls

They Said It

Labels 263 and larger are currently reserved for future extensions. Under many cosmological theories, the labels under 263 are adequate to cover the entire expected life span of the universe; in this case no extensions will be necessary.

—D. J. Bernstein, in a proposal for a new 64-bit time format intended to solve the year 2038 problem (cr.yp.to/proto/tai64.txt)

The National Security Agency remains committed to operating system security research in general and specifically in continuing our research using the security-enhanced Linux prototype. Our relationships with open-source researchers have been very beneficial, and we hope to continue and expand such relationships in the future.

—Grant M. Wagner, Technical Director of the US NSA's Secure Systems Research Office

It's perfectly fine to use the name of your pet or child as a password. However, for the sake of security, make sure the names of all your pets and children contain several non-alphanumeric characters.

—Lore Sjöberg, Brunching.com

Archive Index Issue Table of Contents

Advanced search

From the Editor

Don Marti

Issue #103, November 2002

It's that time of year again, folks. The Readers' Choice awards results are in, and who's your favorite *Linux Journal* columnist? Marcel Gagné, *bien sûr*--the virtual French chef who takes you on explorations of fascinating Linux software (and a little *vin rouge*) every month. Marcel, *nous vous aimons.* Check out your other favorites on page 72.

Even though our journal is (mostly) in English, we see that Linux development for users who speak other languages is just as popular around the world as Marcel's *cuisine* is in our pages. In some areas, we English speakers need to catch up.

In our September 2002 issue, Jon “maddog” Hall mentioned the SAGU Project for university administrative software, headquartered at the public university UNIVATES in the state of Rio Grande do Sul in Brazil. At Linux Istanbul, a conference held as part of the Bilism 2002 computer tradeshow in Istanbul in September, Cesar Brod, who works on SAGU and, among other things, on the GNUTECA free library software, said that the project has already exceeded its goals financially. By developing a homegrown solution, UNIVATES saved $130,000 US on software licenses up front. And the annual $70,000 that would have gone to software upgrades and maintenance is more than enough to pay the development team. The university comes out ahead even before they release any free software at all—contributions from outside, educational value and the benefit of having a custom solution are purely bonuses. UNIVATES actually added free software development to its institution-wide mission statement.

For how many places on earth do shrink-wrap software economics make sense anymore? Also in Istanbul, maddog brought up the economic multiplier effect. Develop locally, whether on your own or as part of a global project, and developers spend money locally. Send the money away, and you'll never see it again.

Look for SAGU, GNUTECA and the other projects hosted on the SourceForge descendant codigolivre.org.br, and you'll find a lot of cool software. The original web screens and docs are all in Portuguese. However, SAGU is in the process of being internationalized: “A Japanese guy who lives in Sweden is doing an English version for schools in South Africa”, Cesar said at the Istanbul event.

There is plenty of information on how to make your software work in different languages in this issue. Check out “Introduction to Internationalization Programming” (page 52) for how to do things the standard, GNU, way. And, “Indian Language Solutions for GNU/Linux” (page 46) covers the status of support for the widely spoken but under-supported languages of that area.

We hear that the skeletal creature on the cover is cute with his hide on, but you'll have to wait for the next movie from Jim Henson's Creature Shop to find out. Meanwhile, learn how Linux can drive their creatures on page 28.

As you might already have heard, I am the new editor in chief of *Linux Journal*. Since *LJ* is already my favorite magazine, I don't plan on changing much. So if there's something you'd like to see in *LJ*, or something you're tired of paying for, please send me some mail: info@linuxjournal.com.

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

Archive Index Issue Table of Contents

Advanced search

Continuing Education

Heather Mead

Issue #103, November 2002

The *Linux Journal* web site provides many learning opportunities.

Parking my car this morning, the Tuesday after Labor Day weekend, I noticed that the neighborhood skate park was empty—the kids are back in school. Besides feeling a little sad that my own back-to-school days are long gone, I thought about all the ways our learning never stops.

Along those lines, an abundance of tutorials have been posted on the *Linux Journal* web site in the past month or two. In the spirit of continuing education, I'll take this opportunity to point out a few of these how-to articles, starting with Jos Hartman's “Building an Office Network from Spare Parts” (www.linuxjournal.com/article/6207). This tutorial outlines how Jos and some other members of his church built their office a new Linux-based network (from donated computer parts) that spans 40 kilometers. When they were finished, they'd built an office LAN, an internet server and a backup/testing server.

If you're new to office suites or have recently switched from StarOffice to OpenOffice, Ralph Krause offers two articles that might save you some time on those daily office tasks. “OpenOffice.org Address Books and Form Letters” (www.linuxjournal.com/article/6269) and “Creating Web Pages with OpenOffice.org” (www.linuxjournal.com/article/6289) explain how to perform tasks such as importing an existing address book, connecting to LDAP servers, filtering an address book with SQL queries to create form letters and creating web pages from the various applications with OpenOffice.

On the other end of the user spectrum, Mike Petullo offered the informative “Amateur Video Production Using Free Software and Linux” (www.linuxjournal.com/article/5817). Using only free software, Mike explains how to digitize analog video sources for storage and manipulation, presents tools for editing video on a computer and offers some options for publishing

digital videos, including the video CD (VCD) format that is compatible with various DVD players.

Finally, for everyone wanting a laptop with Linux pre-installed from the manufacturer, well, keep waiting. Until then, however, Brian C. Lane's article, “Installing Red Hat 7.3 on a Compaq Presario 711CL Laptop” (www.linuxjournal.com/article/6291) will help you put a Linux distro on your laptop—a process he didn't find to be as dreadful as he'd heard. Brian discusses power management and how to make sure your laptop is using the correct kind, employing the acpid dæmon to track power events, drivers for LinModem support and playing encrypted DVDs.

Learning something new is a daily occurrence; be sure to look for new tutorials on the *LJ* site. Articles are available on the site dating back to 1994, and new ones are posted every day.

Heather Mead is senior editor of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #103, November 2002

Our experts answer your technical questions.

Building Custom Packets

Can I create my own IP header, with a new static field of 32 bits, and send it to a destination host?

—Muguntha Kumar, mugunth_kay@rediffmail.com

Sure. Look at sendip for a programming example: www.earth.li/projectpurple/progs/sendip.html.

—Marc Merlin, marc_bts@merlins.org

Die, Process, Die, Die!

What does process state D mean in the output of **ps**? Why doesn't the system allow root to kill processes with state D? What should one do if these processes are consuming a lot of system resources, which need to be released for use by other processes?

—Ankit Doshi, doshiaj@yahoo.com

State D means the process is in uninterruptible sleep—sleeping, waiting for something to happen, but it cannot be interrupted, even by **kill -9**. Sometimes the D process is trying to access remote filesystems that are unmounted or no longer available. If that is the case, use the **soft** option to mount.

—Felipe Barousse Boué, fbarousse@piensa.com

Ignore This Message?

At bootup I get:

```
lib/modules/2.4.9-34/kernel/drivers/crypto/
bcm/bcm5820.o:init_module
Cannot locate memory.
```

I am not able to see any fault. Is this significant?

—George Robertson, grobertson29@earthlink.net

This is the driver module for Broadcom Cryptonet BCM5820. If you do not have this device, you can ignore this error.

—Usman Ansari, usmansansari@yahoo.com

RPM, Please Format Results My Way

I was trying to query all of the installed RPMs on my computer, so I used **rpm -qaR**. However, this gave me a lot of information that isn't user-friendly. I read a section in *Maximum RPM* about the **--queryformat** option. Now, I'm using this:

```
rpm -qaR "%{NAME}[%-8{REQUIRENAME}\n]"
```

The results, though, are exactly the same as those from the previous query. Anybody have any ideas how to make the output readable/usable?

—Joe, joeslapnuts@yahoo.com

You need to include **--qf** or **--queryformat** immediately before your query format string. Use **rpm --querytags** to list all the possible tags in the version of RPM you have.

—Don Marti, info@linuxjournal.com

I Have No Screens and I Must X

I have Red Hat 7.2 on a Gateway PC with an ATI RADEON 8500. I installed XFree86 on a number of systems, but I'm stumped with this one. Initially, I set up /etc/X11/XF86Config-4 to use the RADEON driver and according to the log, it loads that driver as well as the ATI driver and a bunch more. All appears to be fine until the end; then it says:

```
(II) Primary Device is: PCI 01:00:0
(EE) No devices detected.
Fatal server error: no screens found
```

Any ideas?

—Chris Carlson, cwcarlson@cox.net

To rule out configuration errors, try using an /etc/X11/XF86Config-4 generated by Xconfigurator.

—Christopher Wingert, cwingert@qualcomm.com

You may need to upgrade your XFree86 to 4.2 (stock Red Hat 7.2 comes with XFree86 4.1), which includes support for your ATI RADEON. See www.xfree86.org/4.2.0/Status6.html#6.

—Felipe Barousse Boué, fbarousse@piensa.com

Bad Luck or Bad Disk?

I encountered a corrupted / filesystem last June. I tried to repair the filesystem (running **fsck**) but was unable to recover it, so I re-installed Red Hat 7.2. After a month, the same thing happened. Now we are trying to find out if this problem is hardware- or software-related. How can I find out?

—Bing, bingcruz@yahoo.com

If you have a stock Red Hat box (i.e., no custom software), it is unlikely to be the RH distribution. I would suspect your hard drive is about to fail.

—Christopher Wingert, cwingert@qualcomm.com

Are you shutting down correctly? If you hit the power switch while a lot is happening, that could corrupt two installs in two months.

—Don Marti, info@linuxjournal.com

Check your logs, especially /var/log/messages, for any indication of disk or disk controller failures or mishaps. If you're re-installing Linux, run the “check for disk bad blocks” option, which makes a more in-depth check of your disk.

—Felipe Barousse Boué, fbarousse@piensa.com

It's almost certainly the hardware. Other possibilities include an overclocked CPU or motherboard, turning the power off without shutting down, overheating and so on.

—Ben Ford, ben@kalifornia.com

1) Next time you install, make two root partitions, one called / and one called /safe. Copy the content of / to /safe every so often, and make sure LILO or GRUB is able to boot from there. That way you won't be dead in the water if this happens again. 2) It's hard to say if it's hardware or software without more information. Consider switching to a journaling filesystem, such as ext3, so your system can recover more easily if you have problems.

—Marc Merlin, marc_bts@merlins.org

Printer Won't

I am running Red Hat 7.2 and trying to install my HP DeskJet 932C. I have done everything by the book, even tried all the drivers listed, and the printer still does not work. Can anyone tell me what the deal is here? Yes, it is a USB printer.

—Tim Fey, TFey@cfl.rr.com

Try **modprobe printer**, and check out /var/log/messages to see if your printer is detected. You can then use **printtool** to configure it for use with your distribution.

—Christopher Wingert, cwingert@qualcomm.com

Try another USB device to make sure the port you are using with the printer is working. If you are getting any error messages please send them.

—Don Marti, info@linuxjournal.com

You do not say if you're having a problem configuring the USB to talk to the printer or a problem with the actual data format sent to the printer with a working USB connection. If it's the latter, I'd suggest you go to CUPS for the latest drivers for your printer, www.cups.org. You can even generate the proper configuration files for your printer on-line.

—Felipe Barousse Boué, fbarousse@piensa.com

Don't Run This Program Twice

In libc is there a function to call to see if my application is already running?

—Pieter Coetzee, pieter@cosmosc.co.za

The way most programs do this is they store their PID in a file such as /var/run/gdm.pid.

—Marc Merlin, marc_bts@merlins.org

/var/run is writable only by root. If you want to have only one copy of an X application running per user, **gnome-moz-remote** probably does something close to what you want. It looks for an already running copy of Mozilla, and if one is running, it opens the given URL with that copy. Otherwise, it starts a new Mozilla process. Look at cvs.gnome.org/lxr/source//libgnome-2/libgnome/gnome-moz-remote.c.

—Don Marti, info@linuxjournal.com

Archive Index Issue Table of Contents

Advanced search

New Products

Heather Mead

Issue #103, November 2002

CodeWeaver's CrossOver Office Server Edition, Qt 3.1, MontaVista Linux Free Preview Kit and more.

CrossOver Office Server Edition

CodeWeavers announced the release of CrossOver Office Server Edition Version 1.2. The Server Edition allows users to operate Microsoft Windows software, including Office, Internet Explorer, Lotus Notes and Quicken, in a distributed, thin-client environment for both Linux and Solaris. Aimed at helping to standardize desktops on a single system, Server Edition can support hundreds of concurrent users on a single server.

Contact CodeWeavers, Inc., 2550 University Avenue West, Suite 439S, St. Paul, Minnesota 55114, www.codeweavers.com.

Qt 3.1

Qt 3.1 is the newest version of the C++ application framework for multiplatform development, enabling developers to write a single source tree that runs natively on various platforms. Among the additions to version 3.1 is ActiveQt, a framework that allows developers to use Qt to create ActiveX controls and to use Qt applications as ActiveX servers. Qt 3.1 also has new facilities for Motif integration, which lets developers gradually insert Qt-based code into existing Motif applications. Other additions include an improved property editor for the Qt Designer layout tool that can operate on multiple widgets simultaneously, improved multithreading and increased class library assistance.

Contact Trolltech Inc., 3350 Scott Boulevard, Building 55, Suite 2, Santa Clara, California 95054, 408-567-0212, www.trolltech.com.

MontaVista Linux Free Preview Kit

MontaVista is offering a free preview kit that allows embedded application developers to test drive MontaVista Linux Professional Edition Version 2.1. The preview kit is designed to use the same network-based development environment as the Professional Edition and includes the DHCP, NFS and TFTP components. A full MontaVista Linux kernel that can run on 11 processors from six families is included. With the kit, developers can explore the application building and debugging processes, as well as test real-time and networking capabilities of the Pro Edition. The kit can be downloaded from www.mvista.com/previewkit/index.html.

Contact MontaVista Software, 1237 East Arques Avenue, Sunnyvale, California 94085, 408-328-9200, www.mvista.com.

3ware Serial ATA Controller

A family of serial ATA RAID controllers that support open-source drivers and management tools for Red Hat and SuSE is now available from 3ware. The Escalade 8500 series offers terabytes of storage, provides point-to-point communication and delivers full bandwidth to each connected drive. The switched fabric design allows the controller to access all drives simultaneously. Escalade RAID controllers support four, eight or 12 drives, with the 12-drive controller storing up to two terabytes of data.

Contact 3ware, Inc., 701 East Middlefield Road, Suite 300, Mountain View, California 94043, 877-883-9273 (toll-free), www.3ware.com.

Acronis System Utilities

Acronis software announced new versions of three system utilities that operate in either automatic or manual mode, accommodating various user levels. TrueImage 6.0 is a disk-imaging application that allows users to restore all the original settings and data on a system in case of loss. MigrateEasy 6.0 is a dedicated migration tool for hard disk drive to hard disk drive migration. OS Selector 8.0 is a multiboot utility that enables multiple OSes to run on a PC, repartitioning and reorganizing hard disks as needed. All three utilities offer support for the ext3 and Reiser filesystems.

Contact Acronis, 395 Oyster Point Boulevard, Suite 213, San Francisco, California 94080, 888-380-3736 (toll-free), www.acronis.com.

Mobius Axon Wireless System

The Mobius Axon Wireless System from Symbol Technologies is a wireless networking switch and access port system designed to support 802.11b, 802.11a, 802.11g and legacy 802.11 frequency hopping technologies. The 1U rackmountable Mobius Axon Switch manages and delivers wireless traffic through the Mobius Access Ports with application-specific security, including encryption, authentication and virtual LAN (VLAN) technology. The Access Ports can be attached to a desktop, wall or above ceiling tiles. The underlying Linux RTOS will allow third parties to add Bluetooth and GPRS access ports when applications become available.

Contact Symbol Technologies, One Symbol Plaza, Holtsville, New York 11742, 800-722-6234 (toll-free), www.symbol.com.

Argus/PMC Graphics Board

Peritek's new dual-channel PCI Mezzanine Card (PMC) graphics board, the Argus/PMC, supports multiple monitors and dual digital/analog video input channels. The board also supplies an onboard USB 2.0 hub. The high-resolution controller supports independent 2-D, 3-D, OpenGL and DirectX-compatible displays in various OS environments. Each dual-display channel has a Borealis 128-bit graphics controller, and the USB hub supports data rates in excess of 400Mb/s on two channels. Other key features of the Argus/PMC include an onboard 66MHz, 32-bit PCI interface, 16MB SGRAM for each controller, VGA and FCode BIOS support on Channel A, PCI and CompactPCI carriers and dual multi-input audio/video digitizers.

Contact Peritek Corporation, 9975 Business Park Avenue, Suite A, San Diego, California 92131, 800-281-4567 (toll-free), www.peritek.com.

Archive Index Issue Table of Contents

Advanced search