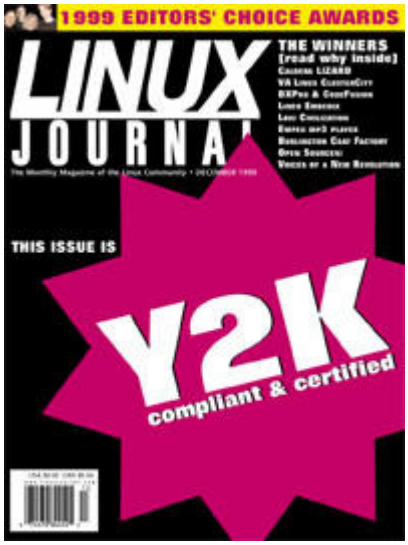
Linux Journal Issue #68/December 1999



Reviews

Columns

Departments

Strictly On-Line

The Army migrates a war game tool from Hewlett Packard 700 series workstations using HP-UX to Intel-based Linux workstations.

Transparent Firewalling *by Federico and Christian Pellegrin*

This article describes how to split an existing network without affecting the configuration of the machines already present by using the proxy arp technique.

Customizing the XDM Login Screen *by Brian Lane*

How would you like your screen to look on start up? Here's how to make it look your way.

Kerberos *by Cosimo Leipold*

Mr. Leipold explains what Kerberos is and why you want to use it.

What Can You Expect?--A Data Collection Project Using Linux *by Denny Fox*

The author describes the end-to-end process of defining and implementing a data collection project using Linux. The project illustrates the use of Expect, stty, cron, a little C programming, gnuplot and ioctl to the serial device.

The Use of Linux in an Embedded System *by Dave Pfaltzgraff*

One company's solution to a customer problem using Linux and open-source software.

Building a Firewall with IP Chains *by Pedro Bueno*

A quick introduction to the program ipchains.

Porting Progress Applications to Linux *by Thomas Barringer*

An explanation of the work required to take an existing Progress application and deploy it on Linux, and the advantages and disadvantages of doing so.

Archive Index

Advanced search

Focus: System Administration

Marjorie Richardson

Issue #68, December 1999

The most important person in any company is most likely the system administrator.

With the increasing dependency on computers by most companies today, the most important person in any company is most likely the system administrator. I know we depend heavily on ours. The hard disk on our server crashed the day we were to start layout last month. Dan Wilder, our sys admin, worked his magic to get us up and running in less than half a day. Without him on site, downtime could conceivably have stretched long enough that we would have missed our print deadlines. His solution was a temporary one but it got us past the critical time, and he has since replaced the disk completely.

Home users are their own system administrators and get to know the fun of logging on as root to give themselves *more power*. Along with power comes responsibility, and keeping the system secure for users—even if only yourself—is a high priority. We have an article this month for the new user about how to make your system secure after installation, as well as one for the more advanced user on keeping the domain name system secure.

We also have an article for company system administrators dealing with integrating Linux in a large-scale network with other systems. Last but not least, we start a two-part series on Virtual Private Networking, a cool way to protect your network. And, of course, don't forget the Strictly On-Line articles (see upFRONT) which include four system administration topics: transparent firewalls, building a firewall with IP chains, Kerberos and Expect.

Awards

We will be attending Comdex about the time this issue is printed in order to give out the Editors' Choice awards and choose the Penguin Playoffs award winners for the Linux Business Expo. Linus Torvalds will be there too, and will

present the Penguin Playoffs awards at the same ceremony. Editors' Choice awards are announced in this issue. Penguin Playoffs and Readers' Choice winners will be announced in the January issue.

—Marjorie Richardson

1999 Editors' Choice Awards

Every year the choices get harder to make with so many exciting new products coming to the Linux world. Already it is getting hard for critics to say “Linux has no applications.” Difficult as it was, we made our decisions and here they are.**by Jason Kroll, Marjorie Richardson, Peter Salus and Doc Searls**

Workings of a Virtual Private Network, Part 1

In this two-part series, Mr. Morgan tells system administrators all about VPNs: describing the technology and discussing the building blocks used in constructing one on a Linux system. VPNs allow you to connect remote networks to the local network, ensuring privacy to both, even through a public link.**by David Morgan**

Post-Installation Security Procedures

When you have finished setting up your system and connecting to the Internet, the next priority is securing your system from outsiders who might want to break in for fun or nefarious purposes. Mr. Harari gives us some straightforward advice on the best ways to secure your system.**by Eddie Harari**

Security Name Servers on UNIX

Security is an important issue for every system, and securing the domain name system is imperative, due to its connection to the Internet. Learn about the vulnerabilities in BIND and how to protect your system from cache poisoning, inverse-query buffer overruns and denial of service. Establish control of your system today.**by Nalneesh Gaur**

Corporate Linux: Coexisting with the Big Boys

Linux has the power to coexist with other systems, UNIX or not. If you are looking to integrate Linux into a large-scale production network, here's how to do it. Mr. Gudjons discusses NIS, NFS and unified login scripts, all necessary to a successful integration.**by Markolf Gudjons**

Archive Index Issue Table of Contents

Advanced search

Workings of a Virtual Private Network, Part 1

David Morgan

Issue #68, December 1999

A look into VPNs—what they are and how they work.

Commercial Virtual Private Network (VPN) products are becoming widespread. They let confidential data safely take the “free ride” offered by the Internet, compensating for the Net's intrinsic lack of security. A Linux VPN can be constructed several ways. One is outlined in the clever but spare VPN mini-HOWTO by Arpad Magosanyi. I implemented it for a business and present some of the insights gained.

Part 1 of this article is theoretical and explanatory, Part 2 practical. I first define VPNs and describe technologies they employ. I discuss the combination of Linux building blocks used by the HOWTO for constructing one. In Part 2, I show log and screen output depicting the results of actually running the script that constructs the VPN.

Virtual Private Networks

A VPN uses a public transport—the Internet—for private communications. It applies encryption to preserve privacy. Traditionally, companies have used private transport to do that—dedicated phone lines. The two ways of keeping an electronic conversation private are to make the line private and the data private. Dedicated lines are private because the line is private, i.e., inaccessible to others. VPNs are private because the data is private, i.e., rendered unintelligible by encryption—different means, same result.

VPNs are most commonly used to connect two networks at different sites of the same company. The technique in effect plugs the remote computers into the local network, consolidating the two physical nets into a single logical one. Remote computers have access to the same local resources as local ones. At the same time, remote machines enjoy the same degree of privacy as local ones. All this is location-transparent in terms of operation (though not

performance) as if they were attached to the local network. This combination of full participation plus full privacy between networks, while using a link that isn't private, is the hallmark of a VPN. The compelling appeal of the VPN is that it's cheap. Dedicated lines are expensive, so displacing them with a free transport is economic.

The Network—PPPD and ROUTE

The VPN in the HOWTO is fashioned from two main ingredients: the secure shell (**ssh**/**sshd**) and the point-to-point protocol (**pppd**). One machine (the “local” one in my terminology, “master” in Mr. Magosanyi's) runs the HOWTO's script to call another (my “remote”, his “slave”). I'll call these VPN servers. The idea is that they belong to the two networks to be joined and serve as the contact points or data conduits between them, on behalf of any remotely situated pair of workstations that want to converse.

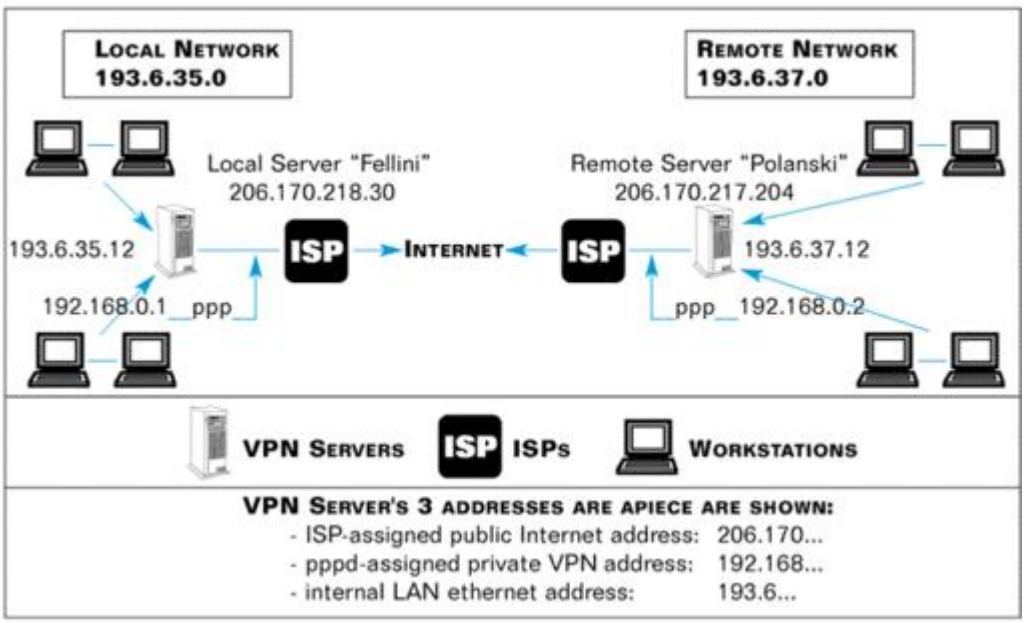


Table 1. VPN Layout

The diagram in Table 1 depicts the layout and addresses in section 4.1 of the VPN HOWTO. For the public Internet addresses (fellini-out, polanski-out), I have substituted those actually in force when I generated the screen captures and log snippets shown later in this article for agreement.

To construct the VPN, the script on the local VPN server must execute four main commands, two of them on the remote VPN server:

- **pppd** remotely, triggered somehow on the other VPN server
- **pppd** locally, on the VPN server where the script runs
- **route** locally
- **route** remotely

The pppd commands establish a working connection. It's strictly a bilateral umbilical cord between the VPN servers that extends no mutual connectivity to workstations on the networks. That is done by the route commands. Once these commands have been executed, the two networks have been transparently pooled into a single group of machines, all mutually visible via Internet addresses.

The Private—SSH

Privacy comes through the tool used by the first computer to trigger commands on the second, because that tool also does authentication and encryption. It's called the secure shell program, which is a remote command executor and an encryptor. Actually, it's a pair of programs, ssh and sshd, deliberately crafted to work together on the client-server model. Other familiar programs that use the model are **ftp/ftpd**, **telnet/telnetd** and any browser/**httpd**.

The “d” in sshd, ftpd and httpd stands for daemon, a synonym for server. Server programs are like genies that grant categories of wishes to their client/ petitioners. So, ftpd grants file wishes to the ftp client and the httpd grants web-page wishes to a browser client. Likewise, sshd grants remote-command wishes to the ssh client. Additionally, ssh and sshd are written to encrypt and decrypt all traffic as it passes between them.

As a command executor, ssh can process a single command and exit. Alternatively, it can set up an open-ended login session where the user submits commands ad hoc. In both cases, ssh delivers back to the local machine the standard output from commands it tells sshd to run in the remote machine. The user sits physically at the local machine, logically logged in and functioning as one of the remote machine's users. All command or session output is delivered from the remote machine to his local monitor. This is much like telnet. Unlike telnet, everything gets encrypted and decrypted on the fly during the session.

The first sentence of the ssh man page highlights these roles:

> ssh (Secure Shell) is a program for logging into a remote machine and for executing commands in a remote machine. It is intended to replace rlogin and rsh, and provide secure encrypted communications between two untrusted hosts over an insecure channel.

The syntax for setting up a remote login session is

```
ssh  -l
```

The syntax for executing a single remote command is

```
ssh  -l
```

-l stands for “login” and specifies the user name for the remote computer login. The first command form gets you logged into the other machine as *remote-user*, with his login prompt on your screen. The second also logs you in and launches *command* on the remote machine all at one stroke. When the command terminates in the latter case, so does your connection. If *command* is **ls /home**, the listing of the other machine's /home subdirectory will be delivered to your screen. Here's an actual screen capture of it:

```
# ssh  -l slave  206.170.217.204  ls /home
david
ftp
httpd
panderson
samba
slave
```

The login prompt is that of the local machine, where the user is seated. The output comes from the remote machine, where **ls** was run (as a user on that machine called slave), but appears here on the local monitor. It shows the contents found in the remote directory /home.

Notice execution was unimpeded by password challenge. This is surprising for a program that's supposed to provide security. However, ssh did authenticate in an alternative, transparent way; its technique uses public-key cryptography and is called RSA authentication. I'll show you the evidence from the remote machine's log file, then explain how the keys work.

Concurrent with the above local activity, these entries appeared in the remote machine's log file (/var/log/messages):

```
Nov  7 20:15:54 localhost sshd[1400]: log: Connection from 206.170.218.30 port 1023
Nov  7 20:15:57 localhost sshd[1400]: log: RSA authentication for slave accepted.
Nov  7 20:15:57 localhost sshd[1402]: log: executing remote command as user slave
Nov  7 20:15:58 localhost sshd[1400]: log: Closing connection to 206.170.218.30
Nov  7 20:15:58 localhost PAM_pwdb[1400]: (ssh) session closed for user slave
```

Note that these entries were authored by sshd, the server half of the secure shell tandem of programs. That's reasonable, since the calling program on the local machine was the client half, ssh. By design, ssh calls and asks for the sshd process, hooking up to it by TCP port number where sshd runs, 22 by default but configurable. **sshd** then swings into action and they proceed to do their thing: authenticate and encrypt. We just saw secondary evidence of the authentication, although so far we've seen no evidence of the encryption. Both rely on the use of keys, in particular the matched pairs of keys that characterize public-key cryptography. Let me describe the essential theory and how you configure ssh with keys, before explaining how authentication results from their use.

First, here's a good nutshell summary from the README.SSH file:

> When started, ssh connects sshd on the server machine, verifies that the server machine really is the machine it wanted to connect, exchanges encryption keys (in a manner which prevents an outside listener from getting the keys), performs authentication using...RSA authentication.... The server then (normally) allocates a pseudo-terminal and starts an interactive shell or user program.

Note that using ssh's **-v** option allows you to watch these activities.

Public/Two-Key Cryptography

Public-key cryptography is the historical successor to secret-key cryptography. I call them two-key and single-key cryptography. Early ciphers used the very same key to decrypt as to encrypt. When sending a recipient your scrambled message, you must somehow also supply him your key to enable unscrambling.

The Achilles heel is that you may not supply the key using the same communications channel you are trying to secure. By presupposition, it needs securing—information on it is available to others. Others' inability to penetrate the scrambled messages you plan to send relies on keeping your key secret from them. But if you send it over that channel to your intended recipient, you are in effect sending it to others too, defeating your purpose. The term “secret-key” for this type of cipher reflects its requirement for keeping the key secret so it can work.

With public-key cryptography, there are two keys: a scrambler and a mathematically corresponding unscrambler. A person never gives out the unscrambler. Instead, he distributes the scrambler. Unlike a secret key, it doesn't unscramble anything. It doesn't possess the ability or value of unscrambling power. Therefore—drum roll—it's okay if it gets publicly intercepted.

Also, the parties—sender/encryptor and recipient/decryptor—reverse roles. The recipient-to-be, not the sender, generates the keys. And he, not the sender, distributes the necessary (scrambler) key to the other person. Security comes from the fact that the “power” key—the unscrambler—reposes from the start with the recipient where it's needed and never needs to travel. Transmission risk is thereby eliminated. Achilles heel solved.

SSH Authentication

Now let's see how ssh uses public-key cryptography for RSA authentication, and how it handles encryption. It has a utility, **ssh-keygen**, that generates matched

key pairs and writes them into disk files. Typically, each user who wishes to use ssh/sshd will generate his own key pair, whether actively by running commands on other computers or passively by having users from other computers log in with his user name to run commands. **ssh-keygen** writes the two key-files into the logged-in user's home directory. The file identity.pub contains the public key suitable for distribution to others and is pure ASCII. The file identity contains the private key to be kept secret. A user runs ssh-keygen only once. Table 2 is the layout of these and some other important ssh files (not all discussed here).

Table 2

User authentication works as an interplay between users' key files. (ssh also offers host authentication, involving /etc/ssh/ssh_host_key, not discussed here.) I'm talking about the two users who are always party to an ssh connection. First, when you run ssh from the local machine, you are already logged into it as somebody. Second, with the **-l** option in your ssh command, you specify some target user on the remote machine as the operator there. I'll call these local-user and remote-user. Another key-related file in each ssh user's home directory is authorized_keys. To succeed, RSA authentication must find a copy of local-user's public key embedded in remote-user's authorized_keys file. This will never happen except deliberately. If I, as the local-user, want to be able to log into your machine as you, I send you my public key. I could send you a copy of my identity.pub as a file, or embed its contents in an e-mail message (since it's pure ASCII and security of key transmittal is unimportant). You, the remote-user, will then place my public key into your authorized_keys file with an editor. Authorization will now succeed when I use ssh to log into your machine as you. Conversely, if I want to let you log into my machine as me, you'll send me your public key and I'll drop it into my authorized_keys file.

Authentication by sshd on the remote machine uses the local-user's public key to encrypt something and ship it back to local machine. Local machine must then prove itself by decrypting and sending back to the remote machine data matching the original. At that point, authentication is complete. **sshd** writes “RSA authentication for *remote-user* accepted” into the remote log (as above), and lets the session or command proceed. For implementation purposes, you simply need to follow the key prepositioning rules when configuring the computers to interact through ssh.

As we noticed earlier, this method doesn't involve any password. It cares only whether the “petitioning” user can convincingly come up with the expected public key and then demonstrate his possession of the matching private one. While it's counterintuitive, I routinely log into remote machines without a password—as root!

ssh offers other authentication methods, password checking among them (ssh is extensive, with many more options in its configuration files). These methods can be used instead of or in combination with RSA authentication. For purposes of a VPN, given that RSA authentication satisfies the test of adequate security for most, using it alone is preferred because of its transparency.

ssh Encryption

Once authenticated, the local user can freely operate as the specified user on the remote machine. There, on his behalf, sshd runs the requested command or shell and sends any standard output back to the local machine, but not before first encrypting it. Direct conversation between the machines is all between ssh and sshd. So, ssh is there on the receiving end, knowing what to do with the incoming data stream (decrypt it) and how (using the agreed key). The same thing happens with reverse traffic, ssh encrypting and sshd decrypting.

You might think the encryption key used on each machine for outbound data would be the public key of the other machine's user. However, for performance reasons, ssh and sshd settle instead on a different, secret-key during their initial negotiation phase, and both use that same key for encrypting the session. While ssh-keygen's public/private keys play the central role in authentication, their role in encryption is solely to impenetrably encrypt the initial exchange of this secret key, overcoming the key exchange weakness in secret key cryptography. For ongoing message encryption, however, the public/private keys are not used. Secret-key algorithms are faster than public/private-key algorithms. The securely exchanged secret key, called the “session key”, is used to encrypt the rest of the session.

The important point is that once the session gets underway, ssh and sshd operate as transparent intermediary processes such that the entire session gets encrypted. Nothing moves between the machines unscrambled, so meaningful interception is impossible.

Blending the Ingredients

Now we can put together our VPN. The trick is to strategically submit a certain command for ssh to launch remotely. That command is pppd, the point-to-point protocol daemon.

We know that during a session, ssh and sshd encrypt the entire dataflow of whatever command(s) they launch as it passes between them. The duration of a session is as long as the command takes to execute. So, for commands that run straightaway to termination like **ls /home**, the session is transient because

the command is transient. Not all commands are this expeditious, for example, an editor or pppd.

```
ssh  -l
```

This command stays up all day—you have to kill it to stop it.

Critical for achieving VPN functionality, pppd is itself a traffic carrier for other programs. This implies that everything passing between two computers via a pppd interface launched under ssh control automatically goes through the encryption mill.

The Virtual

Combined with routing, this bilateral umbilical link broadens into a general-purpose bridge that can carry conversations between any pair of workstations on opposite sides. Routing lets each workstation on one LAN *see* those on the opposite LAN by IP address—one big happy family. At the same time, ssh denies that visibility to the outside world. This is precisely the effect of having all the workstations local. With this setup, you have the equivalent of a single LAN, but because that's not truly what you have, your consolidated network is “virtual”.

What can workstations on opposite LANs do here? Whatever a pair of workstations on the same LAN can—more generally, whatever any machines mutually addressable by IP addresses can. In my experience, examples of actual operations between remote machine pairs on a Linux VPN include:

- Microsoft computers conducting MS peer-to-peer resource sharing.
- A Linux machine serving resources to MS machines by running Samba.
- An MS machine running a terminal emulator on an IBM AIX UNIX machine.
- A Linux or MS machine using TELNET to log into another Linux or UNIX machine.

Interacting machines don't know their conversation is being encrypted for much of its journey. They just launch packets at one another by IP address and let their routing tables figure it out. Upon reaching their VPN server, the routing table there points these packets across the ppp interface operated by ssh. That's where the security comes in; otherwise, it's nothing more than routing as usual.

That's it for the theory. It's virtual. It's private. It's a network. So, I trust you'd agree, it's a virtual private network. Part 2 will cover practical operation of the VPN HOWTO script in detail.

Resources

David Morgan is an independent consultant in Los Angeles and a Computer Science instructor at Santa Monica College. He got serious about Linux in 1998. While waiting for it to enter his life, he earned degrees in physics and business, served in the U.S. Peace Corps as a teacher, held technical and product management positions at Rexon Business Machines, Nantucket Corporation, Computer Associates, and Symantec Corporation. He bicycles, backpacks and cooks. Send him your recipes and VPN experiences. He can be reached at dmorgan1@pacbell.net and currently maintains websites at http://www.geocities.com/Yosemite/Gorge/3645/ and http://skydesign.hypermart.net/.

Archive Index Issue Table of Contents

Advanced search

Corporate Linux: Coexisting with the Big Boys

Markolf Gudjons

Issue #68, December 1999

Integrating Linux into a large-scale production network running SPARCs and Windows.

Linux has come a long way in these past few years, no longer a geek toy and well on its way to being a mainstream operating system. Linus Torvalds, with tongue firmly planted in cheek, is striving for world domination; however, one of the more intriguing strengths of Linux is its friendly and fruitful coexistence with other systems, UNIX or not. In fact, its standards-based approach is one of my favorite ways of distinguishing between it and certain commercial products.

This being said, I would like to present some experiences with integrating Linux machines into a production computer network of over 1000 nodes, divided into about two-thirds SPARCs running Sun Solaris and one-third PCs running one of the variety of software packages emanating from Redmond, WA.

Any large UNIX site usually employs the operating system's easy yet effective mechanisms for maintaining a large number of users working on an equally large number of machines. For Linux to participate in such a network, it needs to be able to participate in or even provide any of these services. At our site, these are:

- NIS: Sun's Network Information Services, formerly called Yellow Pages. This is a well-proven mechanism to distribute any kind of information that can be represented as lists, such as user accounts, passwords and printer definitions.
- NFS: the Network File System. This allows mounting of remote file systems, typically with a mix of static and automatic mounts. The former are configured on a per-machine basis and the latter are distributed via NIS maps.

- Unified login scripts: a carefully set up and maintained web of shell scripts, providing users with the required settings to work with their respective applications. This eliminates the need for each user to hack together her own environment with all the support implications.

Additionally, it is almost self-evident that a Linux box should be able to access networked or server-based printers via the LPR protocol as well as utilize all other communication protocols like HTTP, NNTP, SMTP or FTP. Linux has a well-deserved reputation for being an excellent performer in this respect.

The Information Source: Enabling NIS

A three-step approach lets a Linux machine participate in an NIS domain, beginning with the installation of the necessary software. In the case of the Red Hat 5.*x* distributions, these come in two RPM packages—**ypbind** and **yptools**. The former provides the ypbind executable, which must run on any NIS client as a daemon, and provides communication with the NIS server. The latter contains various NIS-related tools for querying NIS tables (**ypcat**, **ypmatch**, **yppoll**) and maintaining the client configuration (**ypwhich**, **ypset**).

Next, the server-side configuration is modified to let the new NIS client participate in the domain. NIS has only basic security mechanisms, with a common one being the "securenets" list enumerating the networks considered secure to participate in a domain. If your Linux box lives in a different subnet than your other UNIX boxes, make sure that network is present in your server's securenets list (commonly located in /var/yp/securenets on a Sun server).

The final hurdle is the client-side configuration. First of all, the ypbind daemon needs to know the NIS domain name (another security precaution, although a rather fragile one). This is set in the file /etc/yp.conf, together with either an NIS server name or the instruction to broadcast for a server. The file needs to contain only a single line in this format:

```
domain
```

The server bigboy.my.net must have an entry in the hosts database, /etc/hosts.

Now the NIS domain name needs to be set. This can be accomplished via the **domainname *my.NIS.domain*** command. To make this setting persist even after a reboot, the domain name should also be entered into the system's network configuration, in the case of Red Hat: /etc/sysconfig/network:

```
DOMAINNAME=
```

After creating a directory /var/yp/binding for ypbind to store binding information in, ypbind can be started via its script: /etc/rc.d/init.d/ypbind start.

Next, we have to let the system know to actually use NIS to resolve things like hostnames, user IDs and passwords. To do this, edit the file /etc/nsswitch.conf and change the corresponding lines for each service with which you would like to use NIS, e.g.:

```
passwd:     files nis
shadow:     files nis
group:      files nis
hosts:      nis files dns
automount:  files nis
```

In the above examples, the login program trying to authenticate a user will consult /etc/nsswitch.conf, see the sequence **files nis** and look for the information in the respective files. Upon failure, it will query the NIS service for the user's password and shadow entries. If this also fails, the login is denied. The reason to have the entry listed as **files nis** is that the root user is normally not defined in NIS (this is considered a security hole). In the case of a network problem, looking in the local passwd/shadow files first lets root log in without further problems.

This is basically it. Once the file is edited and ypbind is running correctly (verify this by looking for suspicious messages in /var/log/messages and in the corresponding file on the NIS server), your machine is part of the NIS domain. Of course, you can reboot if that makes you feel better; it also allows you to test that the system will come up with the correct configuration.

You can verify ypbind's connection to the server (a “binding” in NIS parlance) using the **ypwhich** command. You can also manually look up information: the command **ypmatch joe passwd** will show Joe's entry from the NIS password map.

The Net: Enabling NFS

Now that NIS is working, let's attend to NFS. Depending on who you listen to, NFS is either the evil beast or the magic bullet to all your user data-related problems. In my opinion, NFS makes a large network with huge amounts of user data easy and transparent to set up, but it comes with a massive performance penalty common to all networked file systems. Count on NFS access being on the order of ten times slower than local hard disk file access. Slow or not, large sites simply can't live without NFS.

That said, setting up an NFS client basically follows the same steps as for the NIS client: software installation, server side configuration and client configuration changes.

NFS requires a kernel built with support for it, presumably as a kernel module, but you can compile it into the kernel itself if you wish. If your kernel does not

yet have NFS support, you need to enable it under “Filesystems”. Go to your kernel source directory (most likely /usr/src/linux) and type **make xconfig** or **make menuconfig**. Obviously, to use NFS, the kernel needs to have network support enabled. After compiling and installing the NFS module, your system has all the software it needs. I'd suggest you install one piece of optional software, though, which is **showmount**. Look for a package called something like nfs*client* on your distribution CD-ROM.

On the NFS server, there is usually a file stating which file systems are exported. Depending on the flavor of UNIX, it can be called /etc/exports (SunOS, Linux, *BSD), /etc/dfs/dfstab (Solaris, other System V variants), or something completely different. An OS-independent way of finding that information is to run the showmount command against the NFS server, e.g., **showmount -e**. This will list the exported file systems and also the machines or groups of machines allowed to mount them.

Large sites usually have a need to manage machines in groups. For example, all users' desktop workstations should be able to mount any of the home directories, whereas only servers might be allowed to mount CDs from a networked jukebox. In NIS, this mechanism is provided by the netgroup map, and chances are the showmount command will list only the netgroups allowed to access specific exports. A sample output would be

```
/home/ftp      (everyone)
/homedesktops
/var/mail      mailservers
```

everyone is a special name denoting every machine, while **desktops** and **mailservers** are netgroups. Executing

```
ypmatch -k desktops netgroup
```

might produce:

```
desktops: penguin, turkey, heron
```

For your Linux machine to be able to access the /home, NFS share requires it to belong to the **desktops** netgroup. Otherwise, the server will deny access.

Once your server lets you in, the last obstacle is advertising the NFS exports to your client. The easiest way to handle this is a permanent mount entry in your /etc/fstab, such as:

```
bigboy:/export/home     /home   nfs     0 0
```

This way, /home would be hard-mounted on each boot. While this approach certainly works very well, it has limitations. At our site, we have a mount point for each user's home directory; e.g., /home/joe for Joe and /home/sue for Sue.

With 1200+ users distributed across ten file servers, hard-mounting each directory would require much housekeeping, and a server replacement or elimination would be a major headache.

Fortunately, there is an elegant way around this, called the **automounter**. This enterprising little daemon watches a set of mount points specified in files for access by the operating system. Once an access is detected, the automount daemon tries to mount the export belonging to the mount point. Other than a slight delay, neither applications nor users notice a difference from a regular mount. As might be expected, the automounter will release (**umount**) a mounted file system after a configurable period of inactivity.

To make use of the automounter, install the autofs package and look at the files it installed in the /etc/auto directory. The first and most important is /etc/auto.master which lists each mount point to be supervised by the automounter and its associated map, usually named /etc/auto.*mountpoint*. Each of these maps follows the basic schema set forth in /etc/auto.misc:

```
d       -fstype=iso9660,ro,user :/dev/cdrom
fd      -fstype=auto,user :/dev/fd0
```

In this example, /misc/cd is mounted with the usual options associated with a CD drive on /misc/cd, whereas the floppy currently in drive /dev/fd0 is mounted on /misc/fd. Note that the mounts will not occur until the directory is accessed, e.g., by doing **ls /misc/cd**, and the automounter will automatically create each of the mount points listed in the file.

“Great”, you say, “now, what's all that got to do with NFS and NIS?” Well, the automount maps are actually lists which can be maintained on the NIS server and distributed to the clients. For example, a typical NIS map named auto.home would look like this:

```
joe     bigboy:/export/home/2/joe
sue     beanbox:/export/home/sue
```

Here, then, is the reason to have the huge number of mount points mentioned earlier. If Joe changes jobs and joins the finance department, his home directory can be moved to beanbox. His new entry would then read:

```
joe     beanbox:/export/home/joe
```

but the mount point on his desktop machine is still /home/joe. In other words, even though he changed to another server, he does not need to adapt any of the environment settings, application data paths or shell scripts he might have. Not convinced? Type **grep $HOME $HOME/.*** to see how many instances of your home path are actually saved everywhere.

If, during NIS configuration, you edited your /etc/nsswitch.conf to contain the line:

```
automount:  files nis
```

the automounter will read its startup files from /etc/auto.master. After that, it will query the NIS server for an NIS map named auto.master and will process the entries accordingly. Thus, the above change for user Joe needs to be made only one time on one system (the NIS master), and it will be known to all clients. No entries to forget, no conflicting client configurations. How's that for efficiency?

Login Scripts: A Uniform Approach

What we've done so far has been largely tech-oriented to let our Linux box be a part of the enterprise network. Login scripts, on the other hand, are to be understood on an administrative level. Sites with only a few users may have no need for them, but if you have to support hundreds or even thousands of users of varying degrees of computer literacy and quite likely in different physical locations, you start to look at the situation differently.

Two of the most widely used shells, tcsh and bash, as well as their precursors csh and sh, utilize a two-step setup procedure. Without going into too much detail, files called .login and .profile are executed on login. Afterwards and on each invocation of a non-login shell (opening a new xterm window), a file called .(t)cshrc or .bash_profile is executed. All of these files reside in the user's home directory; there is also a system default login and profile script (note the missing ".”s) stored in the /etc directory.

When a new user is set up at our site, we give her a default set of .login and .cshrc (the csh variants are the standard shells, but the same could also be done for bash) plus some other files. The only thing that needs to be adjusted is the setting for the default printer in .cshrc:

```
setenv PRINTER
```

Listing 1

An example default .login script is shown in Listing 1. First, the script figures out the primary group (inside the backticks) and loads the variable **$SETUP** with the path to that group's setup files, e.g., /usr/local/etc/dotfiles/finance if the user's primary group is finance. Then, a number of so-called setup files are sourced (included) into the currently running script and the commands in them executed. In the case of the setup.OPENWIN script, it might look like this:

```
setenv OPENWINHOME /usr/openwin
setenv MANPATH ${MANPATH}:/usr/openwin/man
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/usr/openwin/lib
```

These scripts ensure each user gets the same environment settings for her particular group. Finally, the windowing system (in this case, OpenWindows, Sun's version of X) is started. The startup.OPENWIN will not return until the user explicitly logs out of the GUI, at which time execution of this .login resumes. It proceeds to delete some files the user may have left behind and logs the user out of the system.

Again, the beauty of this concept is in its simplicity. If we install a new web browser, we need to change only the central setup file to point to the newly installed version. Upon the next login, each user receives the required settings to start it successfully.

The same concept is also employed for the arrangement of each user's GUI environment. The OpenLook Window Manager, OLWM, as well as most other window managers, comes with an application menu which can be customized to include whichever applications a user might like to access easily. The menu description is stored in a file named ~/.openwin-menu. Again, rather than having everyone create or modify their own menus, this file is merely a link to the central one stored in $SETUP/.openwin-menu. In it, a reference to a private menu stored in $HOME/.openwin-private gives each user an easy chance to add personal items. The central menu files are always carefully maintained to make sure each application works as advertised and are updated each time a new application is brought on-line. Support personnel are grateful they can maneuver a user through the menu by phone while looking at the exact same version of the menu.

Slipping into the Establishment

Listing 2

Integrating a Linux machine into this organization requires the basic setup scripts to differentiate between operating systems if paths are not the same or if some features are not available across operating systems. Since OpenWindows is a proprietary Sun package, a way has to be found for a user to get her OW setup when logging into a Sun box and getting a reasonably similar X11 setup on a Linux box. Wanting the least impact on existing scripts, one good way is to insert the passage shown in Listing 2 into the user's .login. This example first sources all setups that are common across all platforms, like the default HTTP proxy settings, possibly the **NNTPSERVER** variable used by newsreaders and others. Then, a switch statement treats each supported operating system independently. In this case, setup.WORDPROC is executed only for SunOS because we have no word processor for Linux. The

setup.WEBBROWSER script is also called from the $SETUP directory, because it can differentiate between operating systems. This makes sense if you use the same applications across all platforms, e.g., **gcc** and Netscape. The OpenWindows and X11 scripts are platform specific.

Adding support for other systems would be easy to implement in the same way. The “default” statement catches unsupported systems and leaves the user at a shell prompt. Quitting this shell, as well as quitting the GUI in the other cases, will continue .login execution and conveniently log the user out.

Having made our way through the setup scripts, the last obstacle to tackle is the GUI environment. Both X11 and OpenWindows make use of the user's .xinitrc script. Luckily, this is just another shell script and can be treated the same way as .login: add a switch statement to distinguish between operating systems. Generally, this shouldn't be necessary if you take care in setting up the paths correctly and calling whatever X clients are started in .xinitrc *without* full paths. So, rather than having:

```
/usr/bin/panel
/usr/X11R6/bin/xload
/usr/local/bin/wmaker
```

to start two clients and the window manager, it is much more convenient to write:

```
if [ -x panel ]; then
        panel &
fi
xload
wmaker
```

This assumes that both **xload** and WindowMaker are locatable through **$PATH**. The Gnome panel application may or may not be present and will be executed only if found.

Conclusion

Introducing a renegade operating system like Linux into the holy grail of a major company's production network takes a lot of enthusiasm, persuasion and lobbying in addition to a fine feeling for nestling it in as smoothly and unobtrusively as possible. If people take notice without being pointed toward the change, something went wrong.

Without sacrificing any of its inherent flexibility, Linux fits the bill almost perfectly. I always take special pride in demonstrating what Linux can do whenever one of its commercial brethren fails to accomplish something satisfactorily, whether it is related to performance issues, the speed and flexibility of open-source software, or the speed with which the operating

system develops. This benefits the whole company and has led to Tux being a well-liked companion on many a desk in addition to the server rooms. This is a testament to the superiority of this OS and should definitely help Linus toward his ultimate goal after all.

All listings referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue68/3528.tgz.

Markolf Gudjons (mgu@gmx.net) works as a system administrator for Ericsson Eurolab, a subsidiary of communications equipment maker Ericsson A/B. He started out with SCO Xenix in his college days and switched to Linux beginning with kernel 0.96. He looks forward to being able to run it anywhere, anytime on the emerging personal computing platforms. His other hobbies include riding his motorcycle, photography and travel.XX

Archive Index Issue Table of Contents

Advanced search

Post-Installation Security Procedures

Eddie Harari

Issue #68, December 1999

This article discusses a few of the many procedures we must take after the install is done, so that the system will not be trivial to hack.

Installing a Linux system is a relatively easy task. Most of the distributions provide automatic installation tools; these tools take care of the installation procedure from beginning to end. The problem with these automatic installation tools is that they tend to make the wrong assumptions about your system. When it comes to security considerations, these wrong assumptions can cause problems.

Securing Linux is not an easy task. You never know who is trying to gather information from your servers or even from your desktop. Some people do not even try to gather information; they just love it when they bring your web server down or have it show their latest work of art.

The Kernel

The first thing to do after installation is remove unneeded kernel features and services. The Linux kernel has many nice networking features; some of these features are needed by our system, some are not. Here are some kernel networking features we should compile into our kernel: IP-firewalling, Tcp Syn Cookies, Drop Source routed frames. The IP-firewalling option enables the setup of IP access lists from the command line. The Tcp Syn Cookies helps to prevent the known SYN flooding denial-of-service attack. Source-routed frames allow an attacker to bypass the normal routing decisions by specifying the routers the packet should go through within the packet data: this is a very bad idea but is sometimes needed. When compiling the kernel we need to go over all the options and enable only the options needed by our system. If a kernel option is needed we can compile it as a part of the kernel, or we can compile the kernel to support this option as a module. New kernels have new features;

when compiling a new kernel we must make sure we know what each and every option we have enabled does.

Disabling Processes

Listing 1

We can view what processes run in our system by typing **ps aux**. The network sockets that are open in our system, can be displayed by typing **netstat -an**. Listing 1 shows some of the open sockets on a Linux machine.

The fewer services our system gives, the better. The **inetd** process listens on the TCP/UDP sockets specified by its configuration file. The configuration file, /etc/inetd.conf by default, tells inetd what sockets it should open and what processes it should execute once a connection is made on the socket. We should go through all services in the /etc/inetd.conf and disable those that we don't need on our system. The best way to disable a service is to put a comment sign **#** at the beginning of the line that configures the service we want to disable. It is a good idea to comment all /etc/inetd.conf services and use secure services instead. As an example, we can disable TELNET and FTP and enable **ssh** and FTP through ssh.

If we must enable a service, we should configure the service in the most secure way it can be provided. The Linux system has many tools to help an administrator provide services securely.

One of these tools is the IP-Firewalling that the kernel supports. Another tool is the **tcpd**, a program that monitors requests for services on the system. It logs and checks the request, and if all the checks show the client can receive the service, it will open the right service for the client. There are two files that tcpd consults when it checks for authorization: /etc/host.allow and /etc/hosts.deny. To enable tcpd checks before a service begins, we simply tell inetd to run tcpd in the configuration file. Most Linux distributions are configured to run inetd with tcpd for most services by default. Here is a line from an inetd.conf that enables the tcpd checks whenever a TELNET connection request from a client arrives:

```
telnet  stream  tcp     nowait  root\
/usr/sbin/tcpd  /usr/sbin/in.telnetd
```

Whenever a telnet connection request arrives, tcpd is activated by inetd. **tcpd** logs the connection request via the **syslogd** service, then consults the hosts.allow file. If the hosts.allow file contains a match of **telnetd** and the requesting client, the telnet connection is considered authorized and the connection is established. If there is no such line, the /etc/hosts.deny file is

consulted. There must be a line specifing telnetd and client *X* in the /etc/hosts.deny files if we don't want client *X* to be answered by our system.

There are more complicated options available in the tcpd configuration files, such as running shell commands after a certain connection request occurs. For further information on the format of the files, look at Section 5 of the manual for hosts_access and hosts_options.

Another way to disable services run via inetd is to run a shell script instead of the process. Let's have a look at a simple shell script. Please note that it is not always safe to use this method

```
$ cat > /usr/sbin/telnetd.new
#!/bin/sh
echo "Please do not use telnet to this computer.\
Use ssh only if you have the correct public key"
$ chmod +x /usr/sbin/telnetd.new
```

We then edit the /etc/inetd.conf file so it will execute our new script instead of in.telnetd. The line should look like this:

```
telnet stream tcp nowait root /usr/sbin/tcpda\
   telnet.new
```

Send **HUP** signal to inetd , so it will read the new configuration:

```
kill -HUP 'ps -aux | grep inetd |
awk '{print $2}' '
```

Test the new configuration:

```
$ telnet localhost
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Please do not use telnet to this computer. Use ssh only
if you have the correct public key.
```

We must remember that each of the processes run through inetd has its own configuration file. In these configuration files there are security settings that can be activated. We can set up these security settings so the services will get extra security. Take FTP, for example; we can use tcpd to check FTP requests, but as an extra security we can use the ftpaccess file to disable ftp access for root just in case. If our machine has more than one Network Interface Card, we may want to use daemons that enable us to specify on which NIC they should open a socket. This type of setting can't be done with inetd in a trivial way. Some of the FTP daemons enable these settings via the configuration file. There is another method to force a network daemon to listen only on a certain interface. Suppose our machine is connected to network A and network B, and it has two IP addresses: IA and IB. We want telnetd to listen only on IPA, so we write a simple program that opens port 23 on the IPB interface and make this program run just before telnetd. After telnetd is up and running, we can kill our

little program. This won't work with every possible daemon, since certain daemons fail to start unless they can open the socket on each and every NIC on the system.

Networking Applications

Some other networking programs do not run through inetd. These are mostly daemons that run through the rc files at system startup. When not needed, these daemons can be disabled by editing the system startup files. The mount daemon, for example, is the daemon that enables people to mount file systems from our Linux machine. If we want to disable the mount daemon completely, we should edit the rc files so the daemon will not run after the system reboots. If we want to let selected clients in our network work with parts of our file systems, we can run **mountd** with a restricted set of rules that will enforce our policy on the client. This will give the client a limited access to our system.

To configure the mountd restrictions, we should edit the /etc/exports file. The /etc/exports file is the file the mount daemon consults before it gives permissions to the client to mount our local file system. Not only can we limit the clients that can mount our local file systems, but we can also enforce options such as read-only, nosuid and more on the authorized clients.

Another often used program that listens for connections on the network is the **lpd**. The line-printer daemon opens port 515 to listen for connections. We can edit the /etc/hosts.equiv or the /etc/hosts.lpd files to disable and enable the service for some clients. With the *port#* argument, we can tell lpd to listen on a different port than 515; this is a good trick as long as it is not the only step we are taking to secure the service.

The X Window System is a network-based window system that enables other clients to send their application's display to our server. These applications can be dangerous since they can read our keystrokes and view the display of other applications on our X server. If we don't need the X networking support, we can run the X server with the option **-nolisten tcp** set. This option causes the X server to not listen to port 6000, and thus not accept connections from any client. To use this option, simply add it to the *clientargs* variable in the /usr/X11/bin/startx script.

If we need to display output from other machines on our X server, we can use the **xhosts** and **xauth** commands to limit the machines and users that can run applications on our X server. The xhost command is very simple: **xhost +*hostname*** or **xhost -*hostname***.

The **+** sign indicates the client has permission to run applications on our X server, using the **-display server:0.0** option from the command line on the client machine. The **-** sign indicates the client does not have permission to run applications on our server, and if a user on the client machine tries to run an X application on our server, he will get an error message indicating he is not authorized to do so.

Internet Servers

DNS servers must be secured. There is a huge amount of information people can get easily, just by transferring our zone file to their systems. Sometimes our zone files contain the inner network addresses of our systems, router addresses and more.

BIND-8 has many neat security features. The latest version of BIND 8 is 8.2.1, and I recommend upgrading name servers to this version. It contains support of access lists (ACL) for zone query and zone transfers. In the BIND configuration file we can limit the machines that can transfer to each and every zone. One more thing we can do is to put our local network zone, if any, in a secure mode, so that **named** will only answer queries of names belonging to that zone to clients in our local network. There are built-in ACL names such as **any** and **none** which we can use in the named configuration file. One big advantage of the new bind versions is the logging. With version 8 we can tweak the logs to show anything we would like to see. And when it comes to security, the log is a very important issue. Listing 1 is an example of a configuration file allowing only local hosts from network 192.168.1/24 to query all zones; it also allows queries from anywhere on the network to query the outside zones only. One more thing to look at in this named configuration file is that zone transfers are only allowed to two other machines on the network and only for the outside zone.

Listing 2

We can play with these new features of named and disable “dns relaying” by allowing the world to query only zones for which our name server is authoritative, and enabling other queries only from our local networks. This kind of setting will disable the possibility that someone from the Internet will send recursive requests to our server.

Another nice feature in the BIND 8 is that the named can run in a **chrooted** environment; this means that if a hacker exploits the named, it will not have access to all of the file system, but to a very small part of it. To make named run in a chrooted environment, we can use the **-t** option from the command line.

The last thing about the DNS is we can make the name daemon run as a non-root user. This is a very good thing to do as in many other programs as in addition to named. By running a process as root, we actually give the process the permission to do anything in our system; we can accept that as long as the process does only what it was programmed to do in the first place. However, if someone can make this process run arbitrary code, for example, then this arbitrary code will run as root. This means any bug or buffer overflow found in this process can give the hacker a root privilege. Since we don't want to make the hacker's life easier, we can have the named run as a different user.

To accomplish this task, we first add the appropriate user and group to the system. Than we use the **-u** and **-g** options from the command line, to specify *userid* and *groupid* to the named process. [More discussion of "Securing Name Servers on UNIX" can be found in the article of that name in this issue.]

POP, IMAP and Others

The problem with POP, IMAP and some other well-known protocols, such as TELNET and FTP, is the user name and password are sent from the client to the server in clear text. This means someone can tap the communication between the server and the client and get user names and passwords. It is also possible to make a brute force attack on the server trying to guess user names and passwords. We can take care of brute force attacks by running a server that checks for such things. Some POP and IMAP servers close the account after five bad passwords are entered; the account is opened only after a waiting period or it may have to be opened manually. There is an interesting solution to clear text passwords. Some of the services support challenge-response passwords as well as the trivial passwords.

For example, we can get a clear TELNET connection with the SKEY package. The SKEY package gives the user a "One Time Password"; even if someone taps the line and gets the password, he can't use this password again to enter the server. Another tool is **stunnel** which was reviewed by David Bandel in the July 1999 *LJ*. **stunnel** gives the ability to connect from client to server in a secure encrypted way for several purposes, such as SMTP, POP and more.

Sendmail

One could fill a book writing about sendmail security. I would like to mention only a few of many more things about sendmail. The first thing is there are alternatives out there that claim to be much more secure then sendmail. It might be worthwhile to test one of these applications. One more thing about sendmail is that with a very simple program a hacker can try to get many user names from our system by using the VRFY protocol command. The VRFY and the EXPN protocol commands should be disabled in the /etc/sendmail.cf file. To

disable these commands, we should use the following line in the sendmail.cf file:

```
O PrivacyOptions=authwarnings\
noexpn novrfy
```

This option will prevent sendmail from answering to VRFY and EXPN commands. It will also cause sendmail to complain about weak security settings. One last thing I like to do with sendmail is to remove the version number from its **HELO** string, so the version number will not be known to the outside.

Conclusions

Much work needs to be done when it comes to security. We should check every day to see what new hacks have appeared and which software should be upgraded for security reasons. When installing a new application, we should always look at the security settings and set them as tight as possible. It will not make our system 100% cracker proof, but it will make it much harder for the cracker to get into our system.

> Eddie Harari can be reached via e-mail at eddie@sela.co.il.

Eddie Harari (eddie@sela.co.il)

Archive Index Issue Table of Contents

Advanced search

Securing Name Servers on UNIX

Nalneesh Gaur

Issue #68, December 1999

Because the DNS plays such a vital role in the Internet, it is important that this service be protected and secured.

The Domain Name System (DNS) is essential to the functioning of the Internet. The DNS organizes the Internet into distributed hierarchical domains. This hierarchical domain structure provides ease of administration and scalability. It must be kept secure.

In July of 1996, Eugene Kashpureff was able to hijack the www.internic.net (Internic) web site to www.alternic.net (Alternic). As a result, visitors to the Internic net site were directed to the Alternic web site. This was done without authorization from Internic. In late 1997, Kashpureff was arrested in Canada. He pleaded guilty to computer fraud in March of 1998. This incident serves to demonstrate the importance of DNS and the impact that a security attack on the DNS could have on organizations that provide services on the Internet. In today's era of e-commerce and “webification” of everything, DNS security is imperative.

BIND (Berkeley Internet Name Domain) is an implementation of DNS. I will describe here the vulnerabilities discovered in BIND and measures you can take to protect against them. I will assume you are familiar with the workings of the Internet and the DNS architecture.

BIND Versions

Two major BIND versions are available today: BIND version 4.9 and the BIND 8 series. Most new development of BIND continues on the 8 series. The latest BIND, version 8.2.1, was released on June 21, 1999, and is available from http://www.isc.org/. In the 4.9 BIND series, the latest version of BIND is 4.9.7.

BIND is usually available as part of most UNIX-based operating systems. However, vendors tend to be behind in adapting to the latest BIND version. You can determine the version of BIND provided by the vendor of your operating system by checking the system log files.

The Internet Software Consortium (ISC) sponsors the development of BIND. The latest version of BIND provides many new features and security enhancements. Chief among these are full support for negative caching, the ability to run multiple virtual DNS servers, bug fixes from previous versions and performance enhancements. Table 1 compares some of the primary differences between BIND 8 and BIND 4.9.7. The ISC states the following about the two different streams of BIND:

> BIND version 4 is officially deprecated in favor of BIND version 8 and no additional development will be done on BIND version 4, other than for security-related patches.

Table 1

BIND Vulnerabilities and Issues

The risk to a BIND server may arise from a need for a functionality that can leave the BIND server susceptible to attacks, mis-configuration of BIND and vulnerabilities in BIND. The following vulnerabilities/issues in BIND could be exploited.

Cache Poisoning

This vulnerability exists in all versions of BIND prior to version 4.9.6 and version 8.1.1. It allowed an intruder to cause a victim name server to query a remote name server controlled by the intruder. The remote name server would return bogus information to the victim name server. The bogus information would be cached on the victim name server for a period specified by the **TTL** field of the record returned by the remote name server. Very simply, this attack allowed the intruder to point the victim name server's host name IP address mapping to an alternate IP address of the intruder's choice. Eugene Kaspureff used cache poisoning to divert the traffic from www.internic.net to www.alternic.net.

Inverse-Query Buffer Overrun

BIND versions prior to BIND 4.9.7 and BIND 8.1.2 are vulnerable to this. This vulnerability allowed an intruder to gain root-level access on the victim name server, or just cause the server to crash. Earlier versions of BIND allowed the inverse-query feature (see Glossary). Actually, according to the DNS specification, the inverse queries are optional. By default, the servers are not

configured to respond to fake queries. However, BIND 8 can be configured to provide fake responses to inverse queries. It is those servers configured to respond to fake queries that are vulnerable. The inverse-query feature code is disabled (commented out in source code) in BIND versions 4.9.7 and later.

Denial of Service

BIND version 4.9.7 and 8.1.2 perform better bounds checking than the previous versions. The previous BIND version could be exploited to access an invalid memory location causing the server to crash. A crash leaves the name server unable to answer queries, which is a denial of service.

Zone Transfers

Slave name servers perform a zone transfer from the master name server to update their zone database. By default, the master name server will permit zone transfer requests by any host. This does not strictly fall in the category of vulnerabilities. However, the name server contains valuable information about network resources. Information such as the host names, number of hosts, textual information on the hosts (HINFO, TXT) and names of mail servers is made available in zone transfers. Hence, it provides the intruder with intelligence information that can be utilized to launch other types of attacks on an enterprise.

Unauthorized Dynamic Updates

Dynamic updates are associated with BIND versions 8 and later only. Dynamic updates do not apply to the BIND 4 series. The dynamic update feature allows authorized hosts to update the zone records of a name server. If improperly configured, an intruder may be able to add/delete/replace the records for a zone.

Allowing Recursive Queries

This falls more in the category of misuse or abuse of the name server by individuals outside your organization. To put it simply, anyone on the Internet can use your name server to perform recursive queries. This can cause your name server to become extremely busy in responding to everyone else's queries. Additionally, everyone on the Internet will be using your bandwidth to do so. Furthermore, this is related to the cache-poisoning vulnerability.

Securing DNS

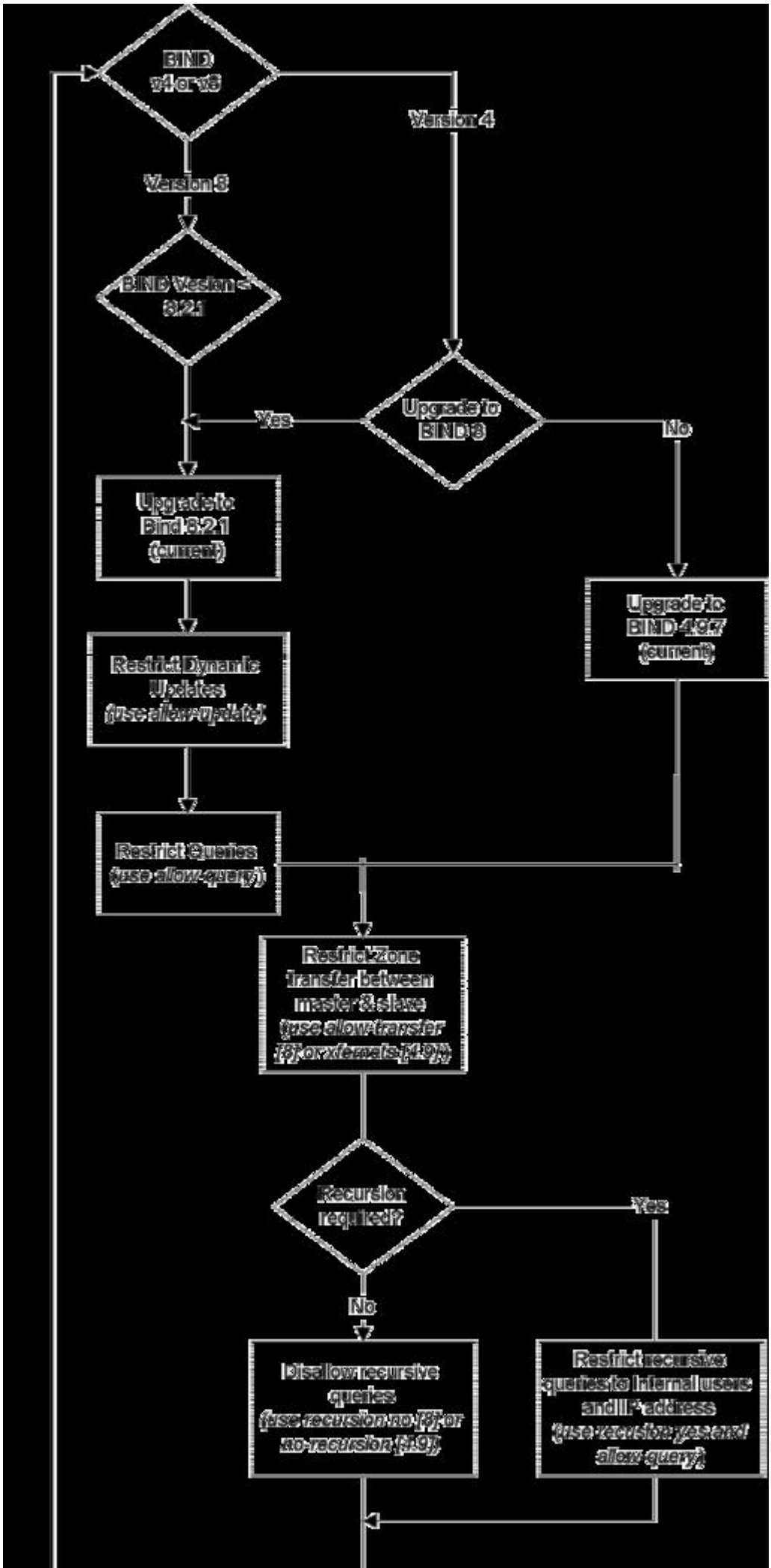

BIND v4 or v8
Version 4
Version 8
BIND Version < 8.2.1
Yes
Upgrade to BIND 8
No
Upgrade to Bind 8.2.1 (current)
Upgrade to BIND 4.9.7 (current)
Restrict Dynamic Updates (use allow-update)
Restrict Queries (use allow-query)
Restrict Zone Transfer between master & slave (use allow-transfer {#;} or xfernets {4.9;})
Recursion required?
Yes
No
Disallow recursive queries (use recursion no; {#;} or no-recursion {4.9;})
Restrict recursive queries to internal users and IP address (use recursion yes; and allow-query;)

Figure 1

This section focuses on measures that name-server administrators can take to secure their DNS environment on UNIX. Figure 1 displays a flow-chart-based approach to securing BIND. The following measures, when implemented properly, will assist in securing BIND.

Use Most Current Version of BIND

The system syslog files contain information about the current version of BIND a system is running. The BIND 8 series provides greater granularity in defining ACLs (access control lists) and configuring the name server. More specifically, BIND 8 series is preferred over BIND 4. Using the most current version of BIND 8 series will protect against the cache poisoning, inverse query buffer overrun and the denial of service vulnerabilities.

Restrict Zone Transfers

BIND provides configuration options to restrict zone transfers. By default, Zone transfer is enabled and anyone can perform zone transfers against the name server database. The **ls {*domain_name*}** command in **nslookup** facilitates this. To restrict zone transfers, use the **allow-transfer** and **xfernets** configuration statements in BIND version 8 and 4.9, respectively.

Establish Access Control on Queries

This is necessary to restrict the hosts that can query the name server. In particular, this is useful for zones internal to an organization. Furthermore, restricting queries minimizes exposure to the cache-poisoning vulnerability. By default, BIND permits anyone to query, even for zones for which a name server is not authoritative. Only BIND 8 provides ACLs for queries. The BIND 8 configuration statement **allow-query** is used to define the ACL for queries based on IP addresses.

Restrict Recursive Queries

If recursion is not desired, it is best to disable it. Such non-recursive servers are responsible for answering queries for the zones for which they are authoritative. In addition, these servers are difficult to spoof because the server does not cache any data. Most often, internal clients send a recursive query to the name server. In such cases, recursion may be desired and must be enabled. Such servers must permit recursion and establish ACLs on queries (**allow-query**).

Restrict Dynamic Updates

Dynamic updates are a feature of BIND 8. By default, BIND 8 disables dynamic updates. If dynamic updates are required, such updates should be restricted to individual IP addresses rather than network addresses. The **allow-update** configuration statement defines the addresses from which a server will accept updates.

Prevent IP Spoofing

BIND statements such as **xfernets**, **allow-transfer**, **allow-query** and **allow-update** statements define ways of restricting many features by IP addresses. This type of configuration does not provide any protection if the IP addresses are spoofed. To protect against IP spoofing, network administrators must establish proper IP spoofing controls on the firewalls, bastion hosts and intrusion detection systems (IDS).

Architectural and Other Considerations

Some other measures that can be used to secure the name servers include the use of split-brain name servers and removing unnecessary information from the DNS database.

Split-Brain Name Server

A split-brain name server consists of two separate name servers. These are quite common in a firewalled environment. Typically, one name server (sometimes referred to as the external name server, outside the firewall) provides information on the web servers, the MX records, other name servers and names associated with any other services offered by a location. The other name server, inside the firewall, contains information on the other name servers in the enterprise. As you can see, the external name server is the interface to the outside world and therefore provides only minimal information. To provide further protection, the external name server can be configured to turn off recursion.

Name Servers with Defined Roles

The idea is to differentiate between name servers that permit recursion and those that do not. It is necessary to permit recursion on some name servers so that clients issuing recursive queries continue to operate. In such situations, one name server can be configured with recursion turned off. Such a name server is authoritative for its zone and will reject recursive queries. Typically, this is the type of server registered with the network domain registries. The other name server permits recursion; however, it permits such queries from

only a few authorized hosts/networks. This allows only authorized resolvers to query the recursive name server.

Remove Unnecessary Information in Database

Unnecessary data in the name server makes it easier to gather intelligence from the name server. Information such as the names of users and administrators, phone numbers and detailed information about the host's make and model, if unnecessary, may be omitted from the name server database.

Run BIND as a Non-Root User

BIND normally runs as root. Running BIND as root may allow an intruder to exploit vulnerabilities, allowing them to run commands and read/write files as root. BIND server 8.1.2 and later provide you with the option to run BIND as a user other than root. In addition, BIND 8 also provides the option to **chroot** the name server. The **u** and **t** options to the name server daemon accomplish this.

Conclusion

Significant security enhancements and improvements have been made in the BIND implementation of DNS. At a minimum, install the most recent version of BIND to protect against commonly known BIND vulnerabilities. To further secure your environment, use the security configuration options in the latest release of BIND. Be careful when you apply BIND updates to your environment, especially if you obtain them from the ISC web site as opposed to your OS vendor. Proper care must be taken when applying vendor patches. Often, vendor patches will overwrite the BIND executable and other related files. The result may be a broken or vulnerable name server.

To keep up with the latest developments in BIND, read the BIND newsgroups, periodically check the ISC web site, and review all DNS-related announcements from your OS vendor. BIND 8.2 includes DNS security features as specified by RFC 2065 (DNSSEC, DNS Security Extensions). The DNS security features use public key cryptography to provide data origin authentication and data integrity. Each zone has its own private/public key and uses its private key to sign the resource records. The DNSSEC extensions are currently not widely deployed. Under DNSSEC, the public key of a zone needs to be signed by its parent zone. As of this writing, the Internic does not yet sign the child zone keys.

Finally, it is most important to say that DNS is only a service, meaning you will be able to access a resource if you have its IP address. Your name server is only as secure as the network and other servers on the network. It is thus

paramount that you have suitable network and server security measures in place before trying to protect your name servers.

Resources

Glossary

Nalneesh Gaur (Nalneesh.Gaur@gte.net) is a manager in the eRisk Solutions practice of Ernst & Young LLP in Dallas, Texas. He has specialized in UNIX and Windows NT systems, integration and Internet/intranet security issues for a number of years.

Archive Index Issue Table of Contents

Advanced search

1999 Editors' Choice Awards

Jason Kroll

Marjorie Richardson

Doc Searls

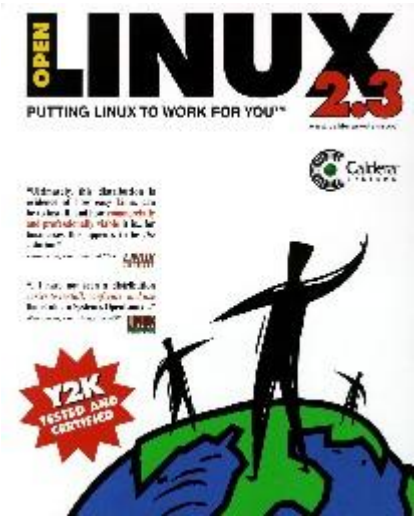
Peter Salus

Issue #68, December 1999

Once again, it is time to present our annual awards to those we feel deserve recognition for their contributions to forwarding the Linux cause in the real world.

Welcome to the 1999 Editors' Choice Awards, brought to you by the same cool cats who bring you *Linux Journal*. This has been an extremely exciting year for the world of Linux in all areas. We have witnessed an explosion in many areas, and the spirit of open source is stronger than ever throughout the entire community. Last year Netscape brought open source to the masses, and we are stronger now in numbers and spirit. This year saw major improvements to the various distributions, a wildly successful IPO, and more attention from the business world and software firms than for other platforms, not to mention the successes of the kernel developers and the proliferation and growth of projects. As for the Editors' Choice Awards, there were many qualified candidates, and while sometimes the winners were obvious, other times we

had to choose the best among peers. Here are our choices, we hope you agree with most (or at least some) of them!



Product of the Year: Caldera Linux 2.3 for LIZARD

Caldera with some help from TrollTech has finally delivered what Linux users and Linux critics have been asking for—easy Linux installation. Not only does LIZARD (LInux wiZARD) make a complete installation with an easy-to-use graphical interface, it automatically probes hardware and installs while you are making configuration decisions about the system. Recent major improvements include automatic detection and support for sound cards, as well as releasing LIZARD as open source under the QPL. LIZARD makes Linux installation practically automatic, setting a new standard across the Linux world. Now Linux is more accessible than ever. And, the Caldera figures who come on during installation are so exciting, we in editorial have been known to install Caldera over and over again on the same machine just to watch what we call “the Ransom dance”. Thank you Caldera and TrollTech.

Best New Hardware: VA Linux Systems ClusterCity & VACM

VA Linux Systems has developed the most advanced Linux Beowulf cluster and requisite management software. ClusterCity is the name of the system, and VACM (pronounced “vacuum”) is the management software released under the

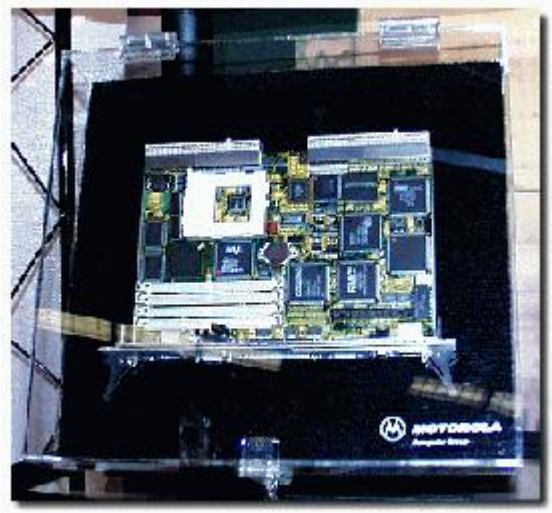
GPL. Not only is the cluster convenient and flexible (rack mounted with easy access to components, as opposed to many clusters which are dozens of desktop boxes stuck together consuming a lot of space), the VACM software represents a major leap forward in cluster management technology. ClusterCity and VACM are to be formally unveiled at SuperComputing 1999. One outstanding (and rather "modern") feature is that VACM allows *complete* remote access to everything, including the BIOS configuration, RAID control, power, system diagnosis, as well as the usual cluster management activities. As Linux storms the server world and marches across the desktop, VA is leading the charge into commercial supercomputing. ClusterCity is already in use by numerous scientific and Internet organizations.

Best New Application—Software Development: BXPro & Code Fusion

This award goes jointly to ICS's BXPro (Builder Xcessory Pro) and Cygnus Code Fusion for their cooperative effort to coordinate their two development products to provide what is essentially the first Visual

C/C++ equivalent for Linux. BXPro and Code Fusion are distribution independent, the former being a fast GUI builder and the latter being a complete C/C++/Java IDE which mostly cooperates with the open-source world of Makefiles. Flexibility and cooperation deserve recognition and help to make top-of-the-line software even better. In fact, one of the rare good qualities of MS products is that they cooperate well with each other (though they tend to be most disagreeable with everything else), so let's see if we can win here too. Even though many are opposed in principle to commercial software, these products set an example for open-source coders—if successful, these products

could become models for future projects. In the meantime, let's see what software gets written with these excellent development applications.



Best New Application—Embedded Systems: Lineo Embedix

Lineo, which recently branched off from Caldera, has delivered Embedix, an embedded Linux OS based on Caldera OpenLinux. Contrary to the trend of making enormous distributions with more packages than the competitors, Lineo concentrates on packing more Linux into smaller packages. The modular kernel is designed for OEMs to have an easy time adding the software they need, and Lineo provides EmbedixSDK as an embedded software development platform for writing and testing software before embedding it. Currently, Lineo and Caldera are working with Motorola, with others to come in the near future. Embedded systems are going to be a major frontier, and one might speculate how far embedded Linux will go? Could it become even more successful as an embedded product than as a desktop OS? Very likely, exciting times are ahead. Check out our October 1999 issue for an interview with Lineo's Lyle Ball (hint: it's the issue with Lyle Ball on the cover with a Motorola box).

Best New Application—End User: Loki's Civilization: Call to Power

Congratulations to Loki Entertainment for its excellent port of the current incarnation of what is probably the ultimate world-domination game. Civilization is the first commercially available, high-quality Linux game (with possible exceptions depending on your point of view), which hopefully will usher in a new era in Linux—the proliferation of games. Nothing makes a better “killer app” than a killer game. Loki has already delivered more ports of top-quality games, including Myth II and Railroad Tycoon and is at work on Heavy Gear II and Heretic II. *Linux Journal* interviewed Scott Draeker (founder of Loki) and Sam Latinga (lead programmer) in the August 1999 issue. “Civilized gentlemen” was the verdict.

Best New Gadget: Empeg Embedded MP3 Player

Empeg Ltd. of the UK has developed an mpeg player for your car, embedded with Linux. Why Linux? Because they “like it a lot, and worship Linus on a daily basis at our own personal shrine,” says their tech page. Empeg uses a StrongARM processor and a big hard drive and has a toolkit for Linux and an interface so that Windows users can load their favorite mpegs (depending on the model, up to 500 albums) into the player. Linux hackers, we're sure, can figure out what else to do with it. The system features 18-bit DACs, 5-band EQ for each of the 4 channels, an FM radio, bass, treble, loudness, balance and fader controls, and gold-plated connectors, among other things. Although the Windows interface is more full featured than for Linux, the developers use the Linux interface. Empeg proves the viability of Linux in small, embedded gadgetry.

Best New Book: *Open Sources: Voices of a New Revolution*

Open source is the revolution, a movement which could restructure and reshape the digital-information age the way the 1960s revamped Western society. Despite its occasional factual inaccuracies (which have a charm all their own), this book collects several essays from the important figures at the head of the GNU/Linux Open Source scene. Although many excellent technical manuals were introduced this year, a comprehensive collection of open-source insights and philosophy means more to the community at large than any one, single technical manual. We are in the midst of a revolution, a period in the history of computers which hopefully we can appreciate before it is over and we look back on it. Open Source is a movement which runs much deeper than many of us appreciate, encompassing those who are concerned with the capability of producing high-quality, powerful software, and those who uphold freedom, community and principles as the core of the philosophy. This book helps us to appreciate the true meaning of our movement.

Best Business Solution: Burlington Coat Factory

Inspired by summer interns who wanted Linux on their workstations, Burlington Coat Factory installed 1,250 Linux machines from Dell in its 264 stores. In spite of popular allegations that Linux is unsupported and has no software, Red Hat is providing the Red Hat Linux distribution and technical support, while Applixware is providing the business software and additional support. Burlington Coat Factory CIO Mike Prince has long been known for his ability to pick up on technologies early; his track record includes adopting UNIX

and Java long before everyone else, for example. Once again, Linux is proving itself in an extremely successful company, with Red Hat and Applixware showing that Linux firms are enthusiastic to provide software and support. More people may come to embrace Linux not only as a moral alternative to proprietary software but as a practical business solution, viable even for businesses that aren't high-powered computing firms crawling with engineers.

Archive Index Issue Table of Contents

Advanced search

X-ISP

Ibrahim Haddad

Issue #68, December 1999

The purpose of this article is to introduce the readers to X-ISP.

Living in the “information highway” era, we must all connect our computers to the Internet to be able to send e-mail, read newsgroups, browse the Web and communicate with the world in general.

Configuring your machine to connect to the Internet via your ISP can be an easy or difficult process, depending on your choice of operating system and tools.

Since Linux is our platform of choice, we are left to select the tools to use. Using X-ISP to configure our machine insures that we can connect our Linux boxes to the Internet in a short time, without any problems and with the advantage of using an X-based interface.

X-ISP is a visual, X11/XForms-based, user-friendly interface to pppd/chat. It offers an X11 dialup networking tool that can also act as a small ISP and phone company (PTT) database manager, and also as a tool to log dialup costs and usage. In addition to that, it provides maximum feedback from the dialing and login phases on a message browser (Figure 1), versatility in interrupting a call in progress, a manual login terminal window as well as call-back and DNS server selection capabilities.

User Interface

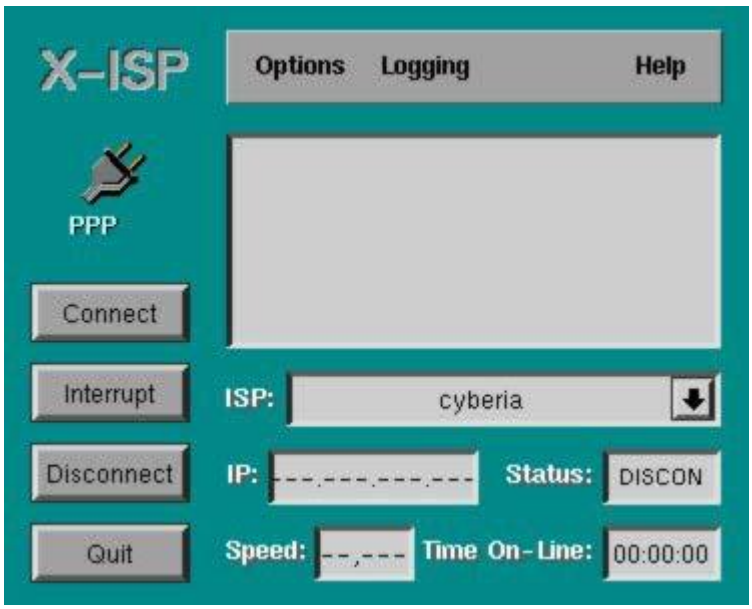


Figure 1. X-ISP Main Window

The user interface of X-ISP is very simple, intuitive and user friendly. It consists of a form with four buttons (Connect, Interrupt, Disconnect and Quit), three menus (Options, Logging and Help) and a drop-choice list of ISP entries (in case the user wants to configure the machine to dial more than one ISP). The "Options" menu contains the five items discussed below.

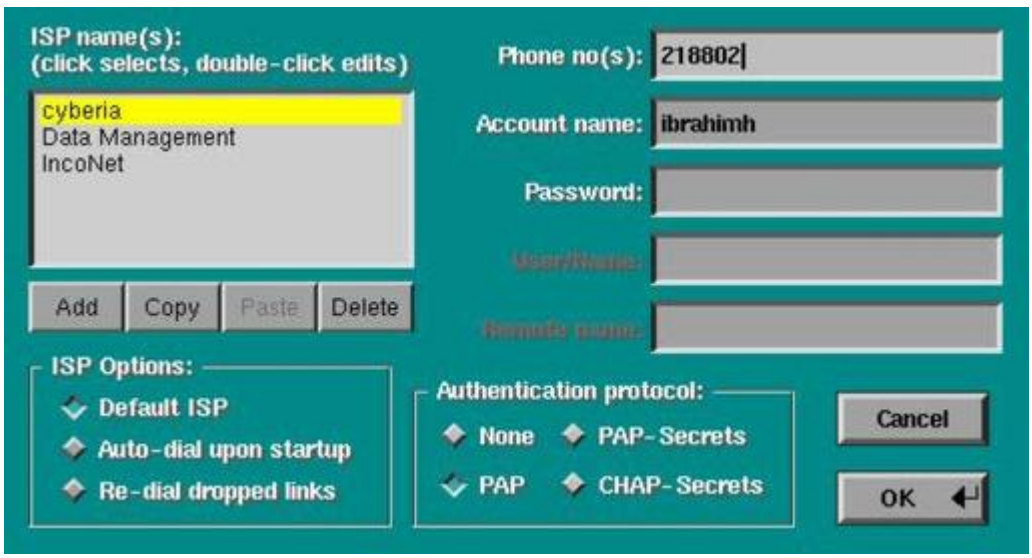


Figure 2. Setting User Account

Information: The user can create ISP entries, set the default ISP, set auto-dial and re-dial on disconnect options and the authentication protocol (none, PAP, PAP-Secrets or CHAT-Secrets). In Figure 2, X-ISP is configured to connect to any of the three ISPs: Cyberia, Data Management or IncoNet (actual ISPs in Lebanon) with Cyberia as the default ISP.

Dialing and Logging: From the dialing and logging window, the user controls several options such as the Dialer Options (number of maximum dialing trials, inter-dialing delay, maximum wait time for connection to be established, etc.), and the Manual and Automatic Logging options. The script section, for both dial-in and call-back, is divided into Expect and Send sections, as used by the call to the chat command. Here, the user must enter the script lines employed by chat to negotiate a successful login for the particular ISP.

Communication Options: These control the settings of the modem device and its properties (Reset and Init strings, baud rates and flow control), dialing method and asyncmap, software compression, serial port baud rate and flow control. All of these options have an initial default value.

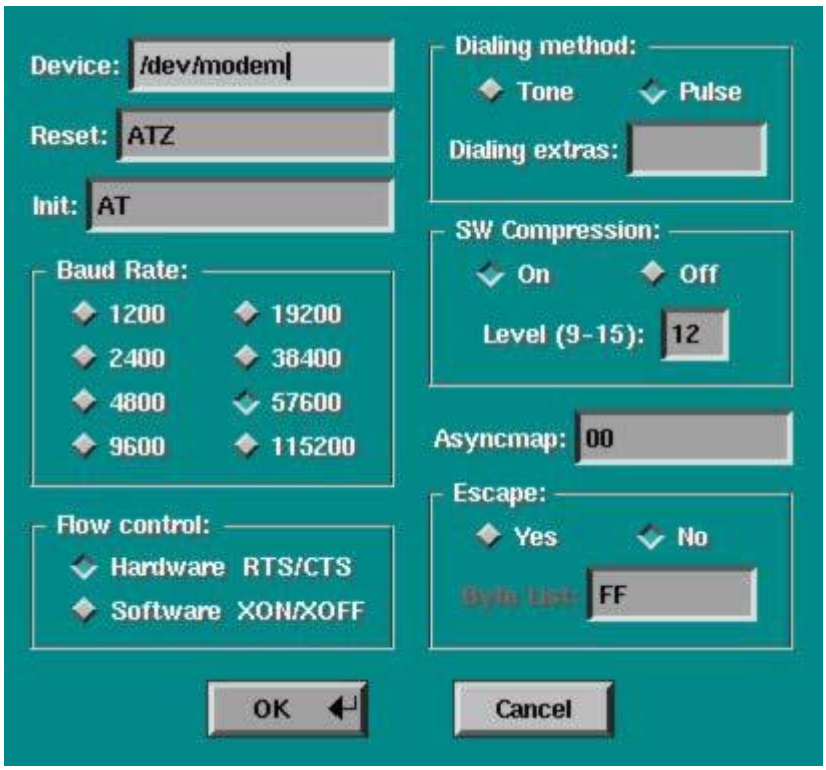


Figure 3. Communication Setup Window

TCP/IP Options: controls the settings of dynamic local and remote addresses, netmask and DNS.

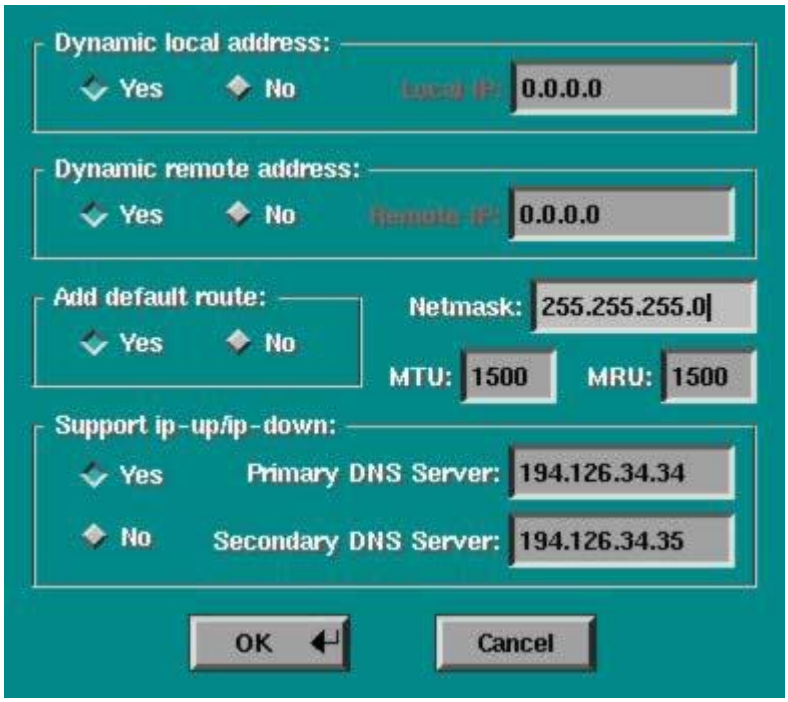


Figure 4. Setting Up the TCP/IP Options

Paths Setup: enables editing the paths to the **pppd** daemon, the location where pppd saves its process ID files, the **chat** utility, the **xispdial** and **xispterm** utilities, and where XISP will keep the named pipe node used for communicating with its components.

Figure 5. Paths Setup

The “Logging” menu contains the following two items:

PTT Editor: This form enables editing of phone company information maintained by X-ISP. The user can add his local phone company to the compiled list and set up its rates. This way, when the user retrieves on-line statistics, he will receive a report of the actual cost.

Statistics: This displays time/cost information and also makes a bar chart of costs for each period (weekly, monthly and bimonthly).

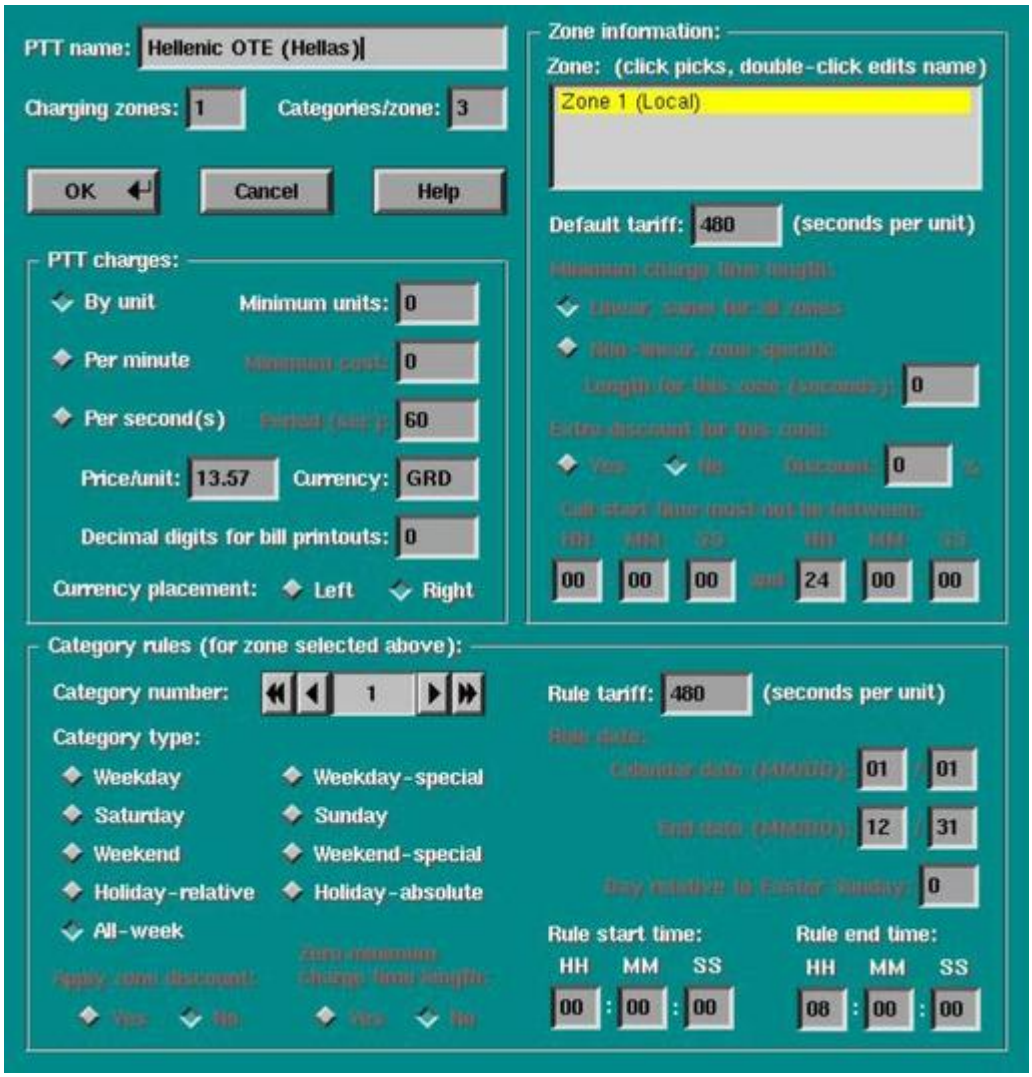


Figure 6. Setting up PTTs

X-ISP Advantages

The X-ISP package implements a user-friendly interface to pppd/chat and provides maximum feedback from the dial-in and login phases on the browser screen. It saves a lot of time compared to configuring PPP manually. X-ISP has several advantages over manual configuration:

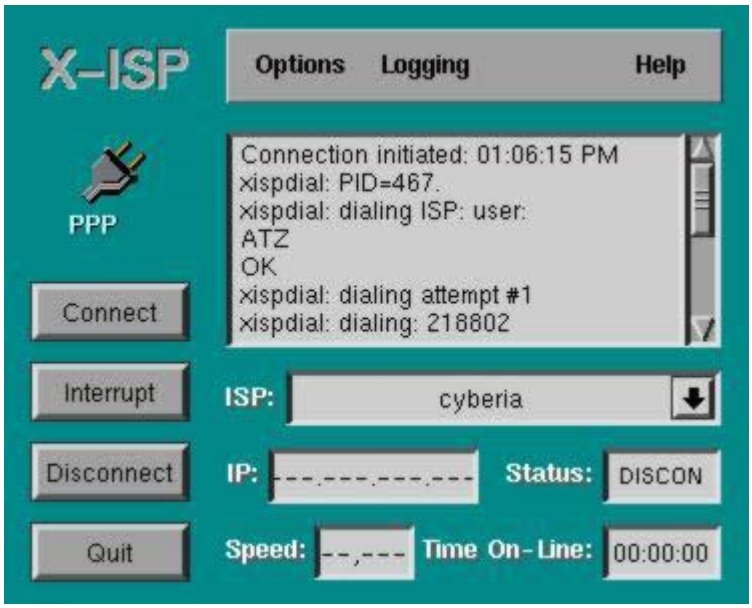


Figure 7. Connection in Process

- X-ISP enhances the user's knowledge of what is happening while a call is in progress in a graphical way, rather than the usual scripts that write output to the terminal.
- X-ISP provides a mechanism to save ISP logs and to keep track of how many calls you make, their duration and cost.
- One major facility is that the user can maintain two databases: one for the ISPs and the other for the phone companies (PTTs). This feature allows the user to configure his machine for more than one ISP and for more than one account with every ISP.
- The phone company database supports all (known) PTT attributes applicable while logging phone-call costs, and saves its information in a separate file in the subdirectory /.xisplogs in the user's home directory.

X-ISP Requirements and Installation

X-ISP was developed by Dimitrios P. Bouras and can be downloaded at no charge from http://users.hol.gr/~dbouras/.

In order to install the X-ISP package on your system, four requirements must be satisfied:

- The ppp-2.2.x package must be installed on the system.
- X11R6 (XFree86 version 3.1.2 or newer) must be installed.
- The xforms libraries (version 0.88 or newer) are needed. You can download a copy from either http://bloch.phys.uwm.edu/xforms/ or http://bragg.phys.uwm.edu/xforms/.
- A copy of the XPM library (version 3.4 or later) is also needed.

Once these requirements are fulfilled, installation is straightforward. I have installed X-ISP on several machines (Slackware 3.4, kernel 2.0.30) by running **make** and **make install**. I was surprised I did not have to re-edit any configuration file or fix file permissions or anything. It worked perfectly from the first trial. However, in case you run into trouble, an explanation on solving your installation problems is in the documentation.

Documentation

X-ISP comes with a large amount of technical documentation discussing implementation issues, security, architecture, and the interaction between the different components. It also has a good help facility that guides the user through setting up X-ISP step by step. Help is also available on-line from the main window.

Final Word

X-ISP is a very well-thought-out tool. It gives us what we need: a fast way to configure the machine, graphical interface, a graphical control over the chat scripts, and a way to tracks time and cost.

Ibrahim F. Haddad is a Ph.D student at Concordia University in Montréal, Canada. Ibrahim got his master's degree from the Lebanese American University (Byblos Campus, Lebanon) where he was first introduced to Linux in 1994. Among his interests are Internet/Intranet and Web development, e-commerce and distributed objects. Ibrahim can be reached via e-mail at ibrahim@ieee.org.

Archive Index Issue Table of Contents

Advanced search

MultiFax

Marcel Gagné

Issue #68, December 1999

Psst! Want to create a Windows broadcast fax system with web-based administration using Linux? Come over here and we'll talk.

Faxing is one of the world's great business constants. This ubiquitous technology has changed the world and can be found in any office and a number of homes around the world. Even when we don't have dedicated fax machines, we use PC or network-based faxing. Putting a fax modem on every PC in your organization doesn't just sound ridiculous—it is. Might I then humbly recommend a great fax solution, free of charge, combined with the dependability of Linux? Wonderful!

While deciding on how to present this article, I thought about all the great free software available for Linux. Your Linux distribution CD contains thousands of such programs. The catch is that not all of them are pretty—extremely functional, but not pretty. Windows users still shudder at the thought of the dreaded command line. So how do we, Linux gurus all, protect them and impress them with the power of our Linux servers? I can hear some of you saying “We don't!”, but seriously, Windows is still out there and will be for some time. Convincing users from the Windows world that hidden behind the command-line interface is a product that will satisfy their needs is sometimes difficult, even when it's something as simple and useful as free network faxing for the office.

As developers and supporters of what will invariably become the next great operating system, both as server and on the desktop, we can ease the confusion of the terrifying command line by wrapping the solutions we offer in friendly, easy-to-use interfaces. My solution often involves hiding the magic of the command line behind a friendly web-style interface. After all, everybody loves their browser—http://www.allplaynowork.com/, anyone?

I will show you how to create an integrated fax solution for your Windows 9*x* users to make faxing as easy as printing. Using the web-based administration tools, you can then off-load some of the responsibility for watching all those faxes go through to your users.

At the end of the article is a URL link from which you can download the full distribution for MultiFax. Find yourself a nice comfy chair, and we'll begin.

Hardware! I Must Have More Hardware!

Well, luckily, not much. You should take a moment and think about this one, though. Will you be using your fax modem as a permanent fax solution, or will you be sharing the fax modem with a dial-on-demand PPP connection? The answer will define whether you can use your current fax modem, or whether you should consider adding another to your system. In my case, I use the same connection for Internet access as for faxes. When I need to pick up a fax (or send one out), I simply take down my Internet connection (by killing my **diald** process), and start a single process for the fax. In a busy office, that may not be such a hot idea.

Nuts and Bolts—mgetty+sendfax

Gert Doering's **mgetty+sendfax** is a relatively easy-to-use fax solution for the office, that is, if you're a command-line-savvy Linux user. The great thing is that it comes with your Linux distribution. Red Hat's package includes it as one of the default choices during installation. I checked my SuSE distribution as well as a Caldera CD and they all had it, so finding it should not be a problem. The latest version of mgetty+sendfax is always available at metalab.unc.edu (formerly sunsite.unc.edu) under the pub/Linux/system/serial/getty directory.

Combined with Horst F.'s **respond** utility for Windows, it provides a rugged and easy-to-use fax mechanism from the Windows desktop. Horst's utility, which lives on your task bar as a tray icon, appears as a fill-in form whenever a user submits a fax to the queue. You enter the destination name and phone number, click OK, and off go your faxes. You can get the respond program at www.boerde.de/~horstfhttp://www.boerde.de/~horstf. While there, make sure you pick up the accompanying **printfax.pl** Perl script and the recommended **smb.conf** entry for Samba.

Create a directory on your Windows PCs (or in a shared drive on the network) and copy RESPOND.EXE there. On my office PCs, I put everything communications-related in a directory called C:\COMM. Then I have subdirectories for my applications. RESPOND.EXE lives in C:\COMM\FAX. It doesn't really matter where it lives, so long as you know where you've put it.

Setting Up sendfax

Once the mgetty+sendfax package is installed on your Linux system (whether with an RPM install or compiled from source), you will need to set up some basic configurations before moving on. On my Red Hat 5.2 system, most of these live in the /etc/mgetty+sendfax directory. Your distribution or install may put them in a different place.

You can get a lot of detail from the mgetty+sendfax FAQ on setting things up, but if you're in a hurry to get faxes moving, here are some of the basics. The config files I took the time to set up are named sendfax.config, mgetty.config, faxrunq.config and faxheader.

In the sendfax.config file, I set up the following parameters:

```
fax-devices ttyS1
fax-id 955-555-5555
ignore-carrier y
```

There are other settings, but I was also in a hurry. Since we have only one fax modem, I set up ttyS1 as my **fax-devices** entry. The **fax-id** is the phone number your fax modem announces to the other fax machine. The third line is to correct a nasty US Robotics problem I had with my fax—it tells the fax program not to hang up between pages of my fax transmission.

On to the faxrunq.config file, essentially, I set only one parameter here. That was:

```
success-send-mail Y
```

Since we have e-mail set up in our office, my Linux system sends us mail when a fax has been successfully sent out. You may or may not want to set this one. If you are setting up large broadcast fax lists, you may not want to get a hundred or more messages confirming each successful fax, or maybe you do. Whatever you decide, consider this carefully before you set up your system. Fear not, though—you can always go back and change it if you find that you made the wrong decision (after you've read your 300 confirmation e-mails).

In my mgetty.config file, I also make only one change:

```
fax-mode 0666
```

More on that change later when I discuss security.

Finally, we have the faxheader file. Mine looks something like this:

```
FAX FROM: **Marcel & Sally** 955-555-5555 TO: @T@
 PAGE: @P@ OF @M@
```

Okay, so my number is not 955-555-5555—you got me there. This defines the identification information that appears at the top of every fax we send out. Pages are numbered for easy reassembly when they are inadvertently dropped on the floor.

You should also make sure that you have a fax.allow and a fax.deny file in the directory. The fax.allow is a text file with a simple list of user IDs to be allowed fax access. The fax.deny file is a list of people who are *not* allowed fax access. Alternatively, if you just want everyone to be able to fax, simply omit the fax.allow file and create an empty fax.deny file. If neither file exists, only root can fax. For a network fax solution, this is not a good idea.

One last thing in our mgetty configuration, and we'll move on to the Windows side of the picture. The default umask for /usr/bin/faxspool is 022. Since faxspool is simply a text file, it's easy to modify the netmask so that network users can delete jobs. In the script, we comment out the **umask 022** default and add the line **umask 000**:

```
# Change umask to 000 so that any user can delete
# an old fax
# umask 022
umask 000
```

Earlier, I mentioned security regarding spooled jobs. Security concerns will arise for some users, but in most offices, the fax machine is a fairly public tool. We look at the network fax queue in much the same way. Anyone can check on faxes, clean up their sent jobs, or print out incoming jobs. The odds of a co-worker deleting your job out of spite is generally fairly remote, at least in our office—I can't guarantee civil behavior in yours. As for outside users, I'll assume your firewall already takes care of them.

Speaking of outside users: would you like to receive faxes as well? That one is easy. The mgetty process can be added to your inittab to listen for incoming calls on your fax modem. Edit your /etc/inittab and add this line:

```
fax1:2345:respawn:/sbin/mgetty /dev/ttyS0
```

In the above entry, fax1 is an arbitrary name I've chosen. In your case, you may pick a different name just as you may need to specify a different device than my ttyS0. Finally, tell the system to reread the inittab with this command:

```
init q
```

mgetty is ready to receive incoming faxes and will restart the listener process each time it hangs up. In my office, we share the fax with our Internet connection, so I simply start a single instance of mgetty with this command:

```
/sbin/mgetty /dev/ttyS0 &
```

The Windows Side of the Picture

Now we want to give our Windows users access to the network fax. A detailed description of setting up Samba services is an article on its own, so I won't cover it here (see Resources). The following snippet from my own smb.conf file can be appended directly to your smb.conf file to create the network fax entry.

```
[netfax]
   comment = Network Fax
   path = /home/samba/faxdir
   read only = No
   guest ok = Yes
   print ok = Yes
   postscript = Yes
   printing = aix
   print command = (/usr/bin/printfax.pl %I %s\
    %U %m; rm %s) &
```

You can now set up a network printer on each of your Windows 95/98 workstations. For a printer type, I use an HP Laserjet 4 PostScript printer and refer to it as Network Fax. I chose the HP Laserjet 4 more or less at random, but any PostScript definition should work.

Next, add a shortcut to the PC's startup folder that points to your RESPOND.EXE program. Remember where you put it? When it runs, RESPOND.EXE will appear as a small rectangular tray icon in your Windows 95 taskbar.

When a user wishes to send a fax from a Windows program, they simply select the Network Fax printer from the list. When they click OK, respond will pop up with a dialogue box similar to the one in Figure 1. Fill in the blanks and click on “OK” to send your fax.

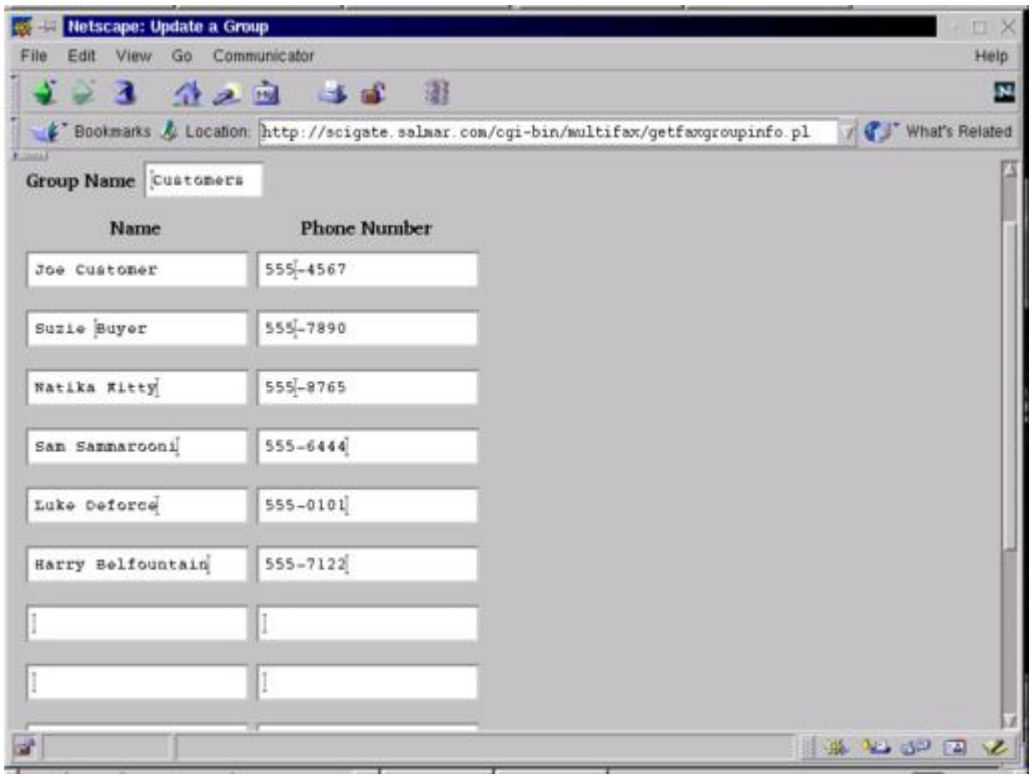


Figure 1. RESPOND Dialogue Image

Now, since part of MultiFax involves the setting up and maintenance of broadcast fax lists, you'll need to do things differently if you are sending to such a list. When prompted for a “Fax Number”, you must enter @ followed by the broadcast fax group name. For example, if the group name is Toronto, the user would enter @Toronto.

If you want your users to get immediate confirmation that their fax job has been spooled (and you probably do), add WINPOPUP.EXE to their startup folders as well. Winpopup comes standard in the Windows 95 distribution and lives in the C:\WINDOWS directory. Then, on the “Winpopup Shortcut” properties tab (accessed with a right-click), I set the **Run:** option to “Minimized”. Winpopup starts up out of the way in the Win95 taskbar, and pops up only when it gets a message. One more thing. Click on “Winpopup” on your taskbar to maximize it. Now click on “Messages”, then “Options”. Click on the checkbox for “Pop up dialog on message receipt”, so that Winpopup pops up each time a message is received.

Winpopup is also a great way for us to send each other little *secret* notes when you're supposed to be working, but I *never* said that.

Installing MultiFax

Now we have mgetty+sendfax ready to go and our Windows PCs all set to fax away. What we want now is a way to report all that activity through a web-browser interface. This is where the MultiFax software comes into play.

The MultiFax administration tool consists of a handful of Perl scripts, web pages and support programs that tie in to mgetty+sendfax. To install MultiFax, follow these steps.

1. Unpack the bundle into a temporary directory using the **tar** command.
2. As root, run the install script by typing **./install**. The install script will do the rest.

Now, the Web Stuff

Administering the queue, monitoring the status of outgoing and incoming faxes, then cleaning up afterward is a little more difficult. You could just have your system delete everything as soon as it is processed, but my experience is that people want a bit more feedback. This leaves us with a cleanup job.

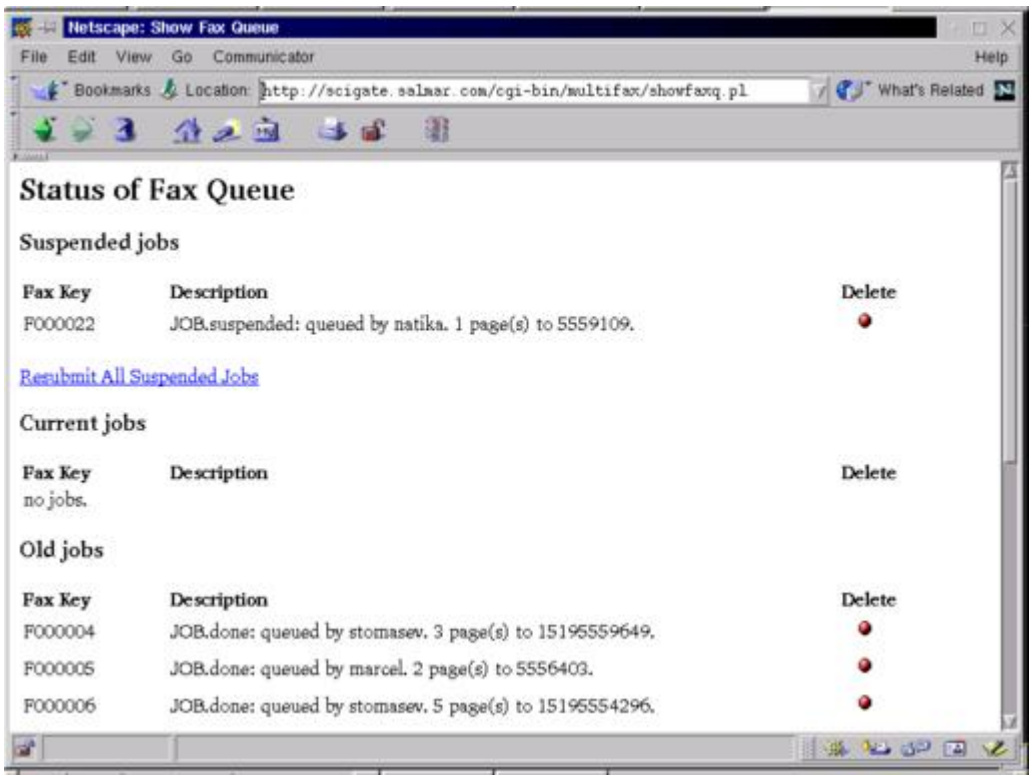


Figure 2. Multifax Main Menu

To access the web tools, point your browser to http://your_serbserver_address/multifax/. You should now be looking at the MultiFax menu (see Figure 2) which contains these four items:

- Check Outgoing Fax Queue Status
- Check Incoming Fax Queue (or print)
- Update Broadcast Fax Groups
- Documentation

The outgoing interface looks at the queue in three different ways: the current outgoing queue, any suspended jobs and successfully sent old jobs. All three views offer the opportunity to delete jobs from the queue. The suspended view has a resend option (when you know beyond a shadow of a doubt that you have the right phone number). See Figure 3 for an example screen.

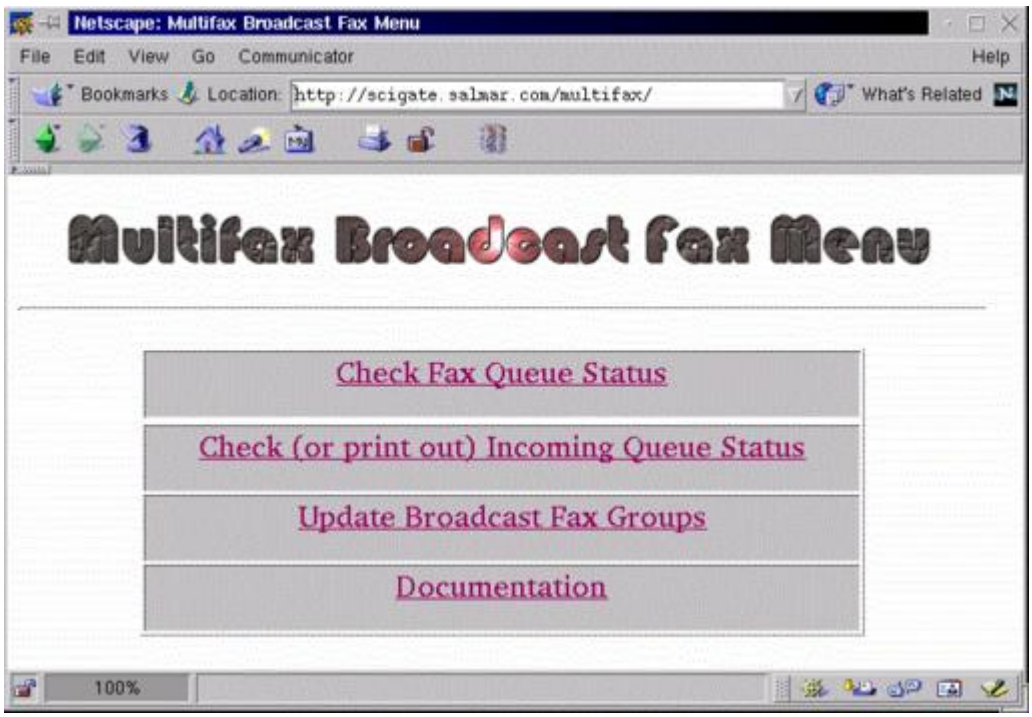


Figure 3. Netscape Screenshot of Fax Queue

The Perl script, **showfaxq.pl**, builds an HTML page by querying the **faxq** for current, suspended, or old (successfully sent) jobs. You have the option at each level to delete a queued job. Strangely enough, the version of **faxrm** included with mgetty does not allow you to delete faxes that have already been sent, only those still waiting to go out. The MultiFax installation will install a modified version of faxrm that takes care of this strange behaviour.

Sending out queued faxes is the job of **faxrunq**, also part of mgetty+sendfax. Processing the queue in this way is probably not what we want to do. There are actually a couple of ways to automate this. The first is to create a **cron** entry that checks the queue and processes it on a regular basis. A good entry for root's crontab would look something like this:

```
0,15,30,45 7-19 * * * /usr/bin/faxrunq -s\
1>/dev/null 2>/dev/null
```

Another option is to run **faxrunqd** which runs as a daemon and regularly checks to see if faxes are waiting to go out. This is by far the easiest way. The cron approach lets you set your own timetable for dealing with the queue.

The next part of the screen shows suspended jobs. Along with the delete option, it is possible to resubmit those suspended jobs to the current job queue.

Finally, we have the old job listing. The only option there is to delete something when you are satisfied that the job has completed and the fax has gone.

Dealing with Incoming Faxes

When faxes arrive, you have the option of printing them immediately using the **new_fax** scripts or simply stacking them up in the queue and printing them manually. In order to have your faxes immediately go to a printer, mgetty+sendfax includes a few scripts to use depending on the eventual destination of your printout. On my system (Red Hat 6.0), the sample scripts are in the /etc/mgetty+sendfax directory. On my local server, we use the **new_fax.lj** script which formats the output for a Laserjet printer. To use the script, simply copy (or rename) the new_fax.*whatever* to new_fax.

The web-based menu lists faxes in the incoming queue and allows for printing or reprinting of any given page. Each entry is identified by date and time, sender and page number. Printing the page is simply a matter of clicking on the red button on the right.

Broadcast Faxing

Now comes the fun part and the real reason a simple web interface for outgoing faxes became so much more. We had many requests for a package that offered broadcast faxing at a decent price (or free). After checking the newsgroups and discovering that those solutions weren't easy to come by, it was obvious we needed to create one.

The mgetty+sendfax package does allow for broadcast faxing, it turns out. As mentioned above, you can specify a list name by sending to @*listname* instead of just a user name. This would require a user to maintain a text list on the Linux server. Not too difficult, but what about our Windows users who would rather not see the shell prompt or deal with **vi**? It is for those users, after all, that we are doing this.

Click on the “Update Broadcast Fax Groups” link from the MultiFax menu. You will be presented with a list of current fax groups. See Figure 4 for an example. You can **add** a new group, **modify** an existing group, or **remove** a group from the list. The basic installation has no groups yet, so you will have only one choice—to add a group.

Figure 4. Broadcast Fax Administration Screen

Let's start by adding a group called “Customers”. Choose “Add a new fax group” from the list, or simply click on the radio button with the same name. Then click on “Submit Request”, and you will be presented with the group update screen. This is the same screen you would see if you chose an existing group and wanted to modify it. The only difference is your group name is blank at this time. Enter **Customers** and tab over to the next field.

Initially, the form has ten rows for names and phone numbers. When you have filled in all ten, you can continue adding more names by clicking the button labelled “Modify an Existing Group”. If you need more than ten, just go back to the broadcast fax menu, select your group (Customers), click on modify, and you will get another ten fields of names to add. In fact, you will always have ten free fields.

Finally, you have the option of removing groups which have become dated or no longer apply. Removing a group from the list starts with the same menu. When you click “Submit Request”, you will be prompted with the confirmation request, “Are you sure you want to do this?” after which the group will be permanently removed.

Documentation? There's Documentation

MultiFax has a fourth menu option with simple, guideline-only documentation. The MultiFax distribution comes with some READMEs and documentation that should answer any other questions that might crop up. Using what's there, you could customize the solution to your own ends.

The Big Wrap-up

Regardless of what you are prepared to spend for a commercial fax solution, there is no such thing as “plug it in and your whole network is up and faxing thirty seconds later”. You, the beleaguered system administrator, will have to do some of your magic to make it happen. Using these instructions, you can create a Linux/Windows network faxing solution that is dependable, inexpensive and fairly simple. Add the web-based fax administration software to that package, and you can unload some of the responsibility of administering network faxing to your users.

Besides, as system administrators, you've got other more important things to worry about—like printers, but we won't go there.

Resources

Marcel Gagné (mggagne@salmar.com) is a longtime Linux user—many different flavours of UNIX, in fact. His company, Salmar Consulting Inc., is a systems integration and network consulting firm specializing in UNIX and TCP/IP networking and the odd Windows NT job. He is also a published science fiction writer with some fantasy tossed in for good measure. When feeling a great need to lose money and time, he plays co-editor and co-publisher of TransVersions, a science fiction, fantasy and horror magazine.

Archive Index Issue Table of Contents

Advanced search

Hell's Kitchen Systems, Inc.

Craig Knudsen

Issue #68, December 1999

Hell's Kitchen Systems, Inc. (HKS) started in 1994 in the Hell's Kitchen neighborhood of Manhattan and moved to Pittsburgh in 1997. Their flagship product is CCVS, a commercial credit card processing system.

HKS is now shipping version 3.2 of CCVS (Credit Card Verification System) and has hundreds of customer sites that include stand-alone merchants, merchant hosts, merchant-application integrators and merchant-application service providers. HKS's goal is to embed an electronic payment processor on every computer.

CCVS uses a computer to perform the same function as a credit-card swipe box found in most retail stores. Acting independently or as a component of a larger system, CCVS can process multiple payment types (credit card, ACH, EFT) in either real time or batch mode.

The system can be used within an electronic storefront on the Internet, or it can help run a mail-order business with custom-built applications for telephone operators.

CCVS can be used within the United States or Canada. It can also be used in other countries with credit-card clearing institutions that support any one of the CCVS-supported protocols.

Currently, CCVS works with either a modem or a leased line to communicate with the same credit-card clearinghouses used by traditional credit-card processing. (HKS plans to support other means of directly contacting clearinghouses, such as TCP/IP.) This approach has a few benefits. There's no need to worry about Internet outages disrupting sales. Additionally, most systems that process credit cards through the Internet (such as CyberCash) charge a per-transaction fee, while HKS charges only for the CCVS software. If

the system is not running on the Internet, there's no need for an Internet connection. This can reduce monthly costs and improve security.

The Linux Connection

HKS first began using Linux in 1995 and now uses it for product development and payment processing for its own customers, as well as testing and demonstration. HKS also makes use of Linux internally for its web server, mail gateway, database server, router, dial-in server and masquerading proxy firewall.

HKS chose Linux as its primary operating system because it liked Linux's versatility, flexibility, open-source code, hardware independence, platform support and low cost. The low cost of Linux allows HKS to run on inexpensive hardware, while compatibility with UNIX systems made Linux an ideal development platform. Access to the Linux kernel source code, especially for serial drivers, made Linux even more attractive. Linux's conformance to the POSIX standard also makes porting to other systems very easy.

Supported Linux Platforms

HKS is committed to supporting as many versions of Linux as possible. This includes distributions from Red Hat, SuSE, Debian, Caldera, Yellow Dog, NetWinder and Cobalt. In addition to Linux, CCVS runs on a variety of other operating systems including BSDI, AIX, FreeBSD, Digital UNIX, SCO OpenServer and SPARC Solaris.

CCVS can be integrated into almost any application because of the wide variety of languages supported. Developers can choose from C, Tcl, Perl5, Python, Java and PHP3.

HKS customers agree that Linux makes good business sense. Approximately 70 to 80% of HKS customers are Linux users (followed by Solaris and FreeBSD users). In fact, many customers choose CCVS because it is the only payment-processing system designed to operate under Linux.

As the first company to develop a commercial credit-card processing system for Linux, HKS is committed to the Open Source movement and plans to sponsor various open-source projects.

HKS provides a downloadable demo of CCVS. Pricing starts at $995 for Linux or OpenBSD and $1295 for commercial UNIX.

Resources

Craig Knudsen (cknudsen@radix.net) lives in Fairfax, VA and telecommutes full-time as a web engineer for ePresence, Inc. of Red Bank, NJ. When he's not working, he and his wife Kim relax with their two Yorkies, Buster and Baloo.

Archive Index Issue Table of Contents

Advanced search

Guido van Rossum

Phil Hughes

Issue #68, December 1999

Phil and Guido stroll through the waterfront at Monterey and discuss Python.

From August 21 through 24, I attended the O'Reilly Open Source Software Convention in Monterey, California. It was a great location, great weather and a great conference. The first two days were tutorials, followed by two days of conferences and the trade show. The show was generally excellent, with over 1600 attendees.

The conference was an outgrowth of a Perl conference, and it had tracks on Linux, Apache, Python, Sendmail and Tcl/Tk, as well as the Perl track. This variety of tracks afforded a great opportunity for the various open-source software tribes to get to know each other.

I tried to play the field and attended a Samba tutorial in the Linux track, a **modperl** tutorial in the Apache track, a Tcl/Tk introduction, and a Python tools for XML programming tutorial.

During the two days of the conference, I spent some time in conferences, including keynotes by Guy Kawasaki and Bill Joy, and some time at the *Linux Journal* booth. As a whole, it was a great conference. Miguel de Iscaza, leader of the GNOME project, summed it up by pointing out that, unlike other recent

shows, this really was a conference for developers by developers. From the tutorials to the included lunches, good organization and good content were evident everywhere. I look forward to attending the conference again next year.

With a large Python track, Guido von Rossum was, of course, in attendance. Our scheduled 30-minute interview turned into an enjoyable three-and-one-half-hour evening on the Monterey waterfront. The first part of the interview is included here. After this interview, Guido went on to talk about CP4E, his project to turn Python into the language used in colleges. That portion of our interview is here: An Interview with Guido van Rossum.

Phil: Python vs. Perl? Obvious question. I heard last night that some Python guys and Perl guys met, and now you are good buddies. Or actually John Orwant said, “The Python guys are a lot nicer than the Perl guys.”

Guido: The funny thing is that while there is a lot of animosity in the lower ranks, I've actually been very friendly with Larry Wall and Tom Christiansen ever since we met five years ago at the VHLL symposium that Tom organized. Tom really wanted me there and made sure I was among the invited speakers describing one of the three major scripting languages.

Phil: What was the third?

Guido: Tcl. Well, maybe Python was the third!

Phil: I've been doing shell programming and AWK programming for close to 20 years and if somebody says, “Here, can you write this in Perl?” the answer is, “Well, yeah I can,” and the reason I can is because I've got 20 years of experience writing in all the pieces that make up Perl. But to go out and tell somebody, “Oh, you are going to learn a new programming language and the design of this language makes perfect sense” is an issue. I don't see how I can do that with Perl, and yet I can do that with Python. That's the distinction for me.

Guido: I think the core of the matter is that Perl has a very UNIX background and claims to be portable. If you talk to people, they all think Windows stinks and UNIX is the one true operating system.

Phil: I think that too!

Guido: Well, you better, given your choice of profession! That is a very typical point of view you find a lot in the Perl world. There also seems to be a bit of dislike for GUIs or anything that smells of user friendly. Part of the UNIX community has that dislike. I grew up in a research environment where we used UNIX; we were one of the earliest UNIX sites in Europe and one of the

earliest Internet sites in Europe as well. But we weren't so partial to UNIX. We were interested in what came next. For a while, I considered myself a graphical-user-interface designer—learned a lot about user interfaces.

Phil: What were you doing at the design end? Or what were you implementating on?

Guido: Oh, the implementation varies—eleven X Window systems as well as Macintosh. I almost never touched a Windows box until Windows 95. Because, before that, any GUI programming was hell. Windows has grown up a lot. There was actually an interesting Python on Windows, even with Windows 3.1. I think that's where Mark Hammon started, actually.

Phil: You said you are not really using Linux right now. What are you using as a platform?

Guido: At work, I have a very big Solaris box on my desk. At home, I have a laptop fast enough and big enough to serve as a desktop machine, and it dual-boots Windows 98 and Linux. I use Windows just for compatibility with many of the people at work and for historical reasons, because we were using a lot of Windows applications. I don't use Windows much for programming; the only programming I do on Windows is—well, there are two kinds—little games programmed in Python, toy applications for testing some idea where it doesn't matter which platform I'm using, and maintenance for the Python port to Windows. The Python port to Windows has two components to it—the core part, which I manage, and the old Windows-specific additions, which Mark Hammon manages. So people typically download two files, and they can get both from http://www.python.org/. The first one is the standard Python distribution which gives you a command-line interpreter, documentation and the standard library—a bunch of goodies. And then, actually, I looked at a bunch of statistics—one third of the people who download Windows installers also download Mark Hammon's extensions.

Phil: What's the ratio of Windows users to UNIX users?

Guido: I don't know about actual users, but downloads from python.org are something like two-thirds Windows and one-third UNIX. All sorts of explanations for that, probably because the UNIX people can build from source. One download may represent a larger number of users. With Windows, I imagine one download often represents one user. Linux, at least, also has the possibility of downloading RPMs from a different site—we don't have any statistics for other sites.

Phil: Plus, Python comes with virtually every Linux distribution anyway.

Guido: Yeah, well, there are lots of complaints about that. They're not very up-to-date and they don't have a very large set of standard extensions, so it takes a little extra work to configure them at compile time. If you want the full Python installation on your Red Hat Linux, for example, you want to download all of Andre's stuff.

Phil: By the way, I actually do use Python. I've been telling people at the office, “We've got to start programming in Python.” We went through a couple of Perl programmers who could produce code extremely fast, but when they found new jobs, we were sitting around going, “What is this stuff?”

Guido: That is one of the things I've always had as part of my philosophy: it's much more important that you can read someone else's code than how fast you can produce your own code. Again, that is something very few people in the Perl community appreciate. I know Tom Christiansen does appreciate it. But many of the people who are in the trenches doing Perl programming don't have the time to learn how to write it right; they just copy the examples they found in the legacy of the previous person who was hacking at that site.

Phil: Sometimes that's fine. I write a lot of disposable stuff in AWK. Here is some data, I've got to turn it into something and load it—fine.

Guido: As long as you actually dispose of it!

Phil: Exactly!

Guido: My dad used to say there is nothing more permanent than a temporary solution. A lot of disposable software ends up persisting beyond all reason.

That is actually one of the big differences between the Perl philosophy and the Python philosophy. Perl gives you lots of different ways of doing things which makes it much harder for the reader, because if you are writing code, you need to know only what you know—you need to know only one way to do what you want to do. But, if you are reading, you need to know everything before you can

actually read it. I mean, it's not so different from understanding a language like English. My passive vocabulary, what I can understand when people speak to me, is a lot larger than my active vocabulary—the words I can actually use myself. Not growing up as a native speaker, my active vocabulary is smaller than that of a typical native speaker. For the casual programmer, you have the same situation where you know enough of the language to get by for your own programming needs. But if someone else presents you with something he wrote, he probably knows a different subset, and there is no agreement on what the basic 600 words are that you both need to know to get by. It's often quicker to run a program and see the results, then reproduce the code in an idiom you know and start hacking on corrections, than it is to try to understand how the code works and change that.

Phil: That's similar to what IBM did with PL/1, the ultimate language. If you went into a company that had both FORTRAN and Cobol programmers and you said, “You are all going to write in PL1,” then the Cobol programmers wrote Cobol in PL/1 and the FORTRAN programmers wrote FORTRAN in PL/1!

Guido: And the Algol programmers wrote Algol in PL/1.

Phil: It seems like Python is starting to be taken really seriously in web development and so on. Is Python being taken seriously in academia? I guess I mean relative to Perl, because Perl isn't, as far as I can see.

Guido: I would say Python is being taken a lot more seriously. There are language designers who don't approve of certain short cuts, or the fact that Python doesn't have static typing, or the fact that there are other languages out there that are as good as Python is, and again borrow all the good features from those languages.

Phil: What languages?

Guido: Some people think, for instance, that Dylan—which I think has a very academic flavor—is everything Python is plus so much more.

Phil: Dylan? I've never heard of it.

Guido: Well, that's exactly Dylan's problem. I don't know, but I think it started out as a LISP variant, with sort of an alternative syntax. The syntax was deliberately unLISPish in order not to scare off everyone who is not already brainwashed with LISP, because LISP has one of the biggest image problems of any programming language in the world.

Phil: I had to learn LISP in college, and I can appreciate that! Keypunching parentheses is not my favorite thing.

Guido: I like a lot of the concepts of LISP, but I strongly disagree with their approach to syntax, which happens to be the same approach Tcl has, more or less. Which is, there is no syntax, or it's so simple you have to do everything else outside the syntax.

Phil: Is Tcl making it at all in academia?

Guido: I am sort of removed from academia, so I don't quite know. I don't think so. I mean, one or two people I spoke to recently from academia had a very strong opinion that Python was a decent language and Perl and Tcl were not.

Phil: To me, the biggest thing lacking in Python is books. To learn the language, basically, there are three books out there: the two O'Reilly books and the one you wrote...

Guido: I didn't write that!

Phil: Okay, the book you didn't write!

Guido: Aaron Watters wrote it. I was originally supposed to write one chapter. When I came back from vacation ready to write the chapter, they had changed the deadline and already started printing! So I thought, hmmm, good! Until the beginning of this year, there were clearly too few books. Right now, *Learning Python* is by far my favorite Python book.

Phil: Hey! It's mine too!

Guido: Yeah. I think what we'll see is that several other publishers will start publishing Python books. I mean, I've been talking to authors, and I actually managed to piss one off because I didn't like the chapter he sent me. But everyone else I'm fairly okay with. And the one I pissed off is going to continue anyway, so more power to him. There will probably be five or six more books by the end of the year. Mark Hammon and Andy Robinson have almost completed another O'Reilly Python book, specifically for Windows users. Dave Beazley, who is solidly in academia—an assistant professor at the University of Chicago—has already finished writing and turned in the manuscript for a quick Python book for the experienced programmer. It has all the information you need to learn the language quickly and start using it. There is one big appendix in the book which is almost the entire library reference manual. All the relevant things are in the New Riders series. I think two different people are working on Python books with Tkinter. There is another quick Python introduction, and plans for something like an 800-page book with lots of annotated examples. So there is lots of stuff coming out, and it's aimed at a much larger variety of users.

The first books that come out try to be everything for everyone, and that is kind of hard.

Phil: I guess the big thing you are playing with now is JPython?

Guido: JPython is really big, but I'm not personally very involved, actually. One of my well-respected colleagues at the CNRI (Corporation for National Research Initiatives) is now doing the maintenance after Jim Hugunin (see http://www.python.org/) and the original author have decided to move to the West Coast.

Phil: JPython is a really neat thing. If Java actually becomes the success it wishes to be ...

Guido: Java is a decent language, but it really needs a companion like Python. I have to admit that was one of the things I was very skeptical about initially when Jim Hugunin came up with the idea. I thought, “Oh, my! That is going to be so slow! That's not going to be worth it.” It was a little slow, but not so slow as to be unusable. I had to push Jim to add some psychological tricks to the way the interpreter initializes itself. I think the current version quickly prints the command line, the first prompt and then actually finishes initialization. This has the advantage that during the second or so it takes you to type your first command, the initialization finishes, so when you hit return, you have an instant response.

Phil: I worked for a computer sciences department on the development of an operating system in 1970 when everyone was using a teletype. We had a basic interpreter, and when you typed **run**, the basic interpreter printed out the name of the program and the current date and time. It did that to cover up the fact that it was compiling it into byte code before it was going to run. Same story, but you had a lot more time when you had about 50 characters to print on a 10-character-per-second teletype.

Guido: Well, that kind of trick still works, but JPython is its own world. I think it is the only language integrated with Java in this particular way, where you basically have cross-language inheritance. You can inherit in Java from a Python class and vice versa, and you can override methods in each direction. So the people who use it, use it typically as an extension language for a large system totally or almost totally written in Java, where they need to provide some end-user program ability. I think one product, Object Domain, is an application that is sort of UML for Java programs, and you can do all sorts of manipulations with the UML objects. In a little corner of the screen is a Python prompt—when what you can do with the menus doesn't give you enough flexibility, you can write your own Python code right there and execute it. That is how JPython is used as an extension language to Java. That is the equivalent of what people have been doing with CPython in C or C++ applications all this time. But with Java, there really weren't very many options like that, and Javascript hasn't made it outside the embedded HTML world.

Phil: I haven't done much with Javascript.

Guido: It's actually a decent language. It doesn't have the whole object-oriented kit and kaboodle Python has, and that's its big weakness. Actually, Javascript and Python share a lot of syntactical ideas, and the dynamics of the language is also more or less the same. It's probably purely coincidental.

Phil: In the near future, we will be developing some software that needs a local user as well as a web-based human interface and a database back end. The local user interface needs to be semi-graphical with minimal keystrokes. We are considering using Python and a database such as MySQL or Postgres. Are we on the right track, or are we crazy?

Guido: There are tons of options. I don't know if you've done any research yet in that area, but I still like Tkinter best—it seems to be the most popular. It is called the de facto standard. I like it because it is portable between Windows, UNIX and Macintosh. I think the only other option that has those three platforms is WX Windows, which is a big C++-based thing. It used to have a Python port that was fairly poor; I think it's all been redone, and I hear it is much better now. I've never used it myself, because I am so happy with Tk interface. I've also seen other good things. Some of my colleagues have just started playing with GTK, and it certainly looks very slick; the code is about as clean as Tkinter code. Apparently, the GTK port for Python is actively being maintained, unlike some other offerings. I think there is a KDE or a Qt port that is not being maintained any longer.

Phil: The biggest problem we decided we had with most Tk documentation is that it is attached to Tcl, which is the best way to confuse somebody who is trying to write something else. I guess there is a book coming out.

Guido: There are already two books coming out—I have no idea when exactly, so it may even be another year. I expect it to be a little earlier than that. In the meantime, the best place to go for information is the HOWTO section, or there is some kind of resource guide on the python.org site where Tkinter is actually one of the topics which basically has pointers to all the other stuff you need to know. The best documentation for Tkinter is the web pages Fredrick Lund put together and had on his Python web site.

When we designed Tkinter, we were aware of the fact that we couldn't possibly document it all. Certainly not document all the semantics. STK was changing because when we started Tkinter, Tcl/Tk was at release 7.3 or so. We knew Tcl/Tk was going to evolve, so we ported most parts. We set up a very regular structure of mapping between Tk commands and Tkinter, mapped in classes. So, you can prepare a small set of rules that say, if it looks like *this* in Tk, it looks like *this* in Python. Along with that, there is a lot of information you can learn from the standard Tcl/Tk documentation. Tcl/Tk has excellent man pages. Whenever you are unsure what options to the **grep** command are or how to do a button that has a particular property, look it up in the Tk documentation and apply the mapping—sure enough, it works in the Python world.

Phil: That's cool. It's likely that's the way we will go.

Guido: Can I offer one piece of advice? I think one of the reasons for the success of Python, Perl, Tcl, Linux and Apache is lots of extensibility. People can scratch their own itch by writing their own little module that does what they need to do so they don't have to bother the original developers with “Can you please add this feature?,” the problem every sort of *fixed* system ends up with.

Phil: I agree completely. There are just too many pieces of software that were written to solve somebody's problem but actually didn't solve it, where the marketing department took care of explaining why it did solve it. There is a company in Redmond that is really known for doing exactly that.

Guido: Their recent offerings are actually pretty extendable, but where they aren't—that's when it's bad.

Phil: Is the Python community ready for a magazine?

Guido: Probably not yet.

Phil: That's what I told John Orwant about the Perl community, but he didn't believe me!

Guido: Well, that's good! With this kind of stuff, I'm always happy to be proven wrong. We tried an on-line magazine and there were one or two issues and that was it.

Phil: Yeah, I saw it.

Guido: Realistically, the Python community is an order of magnitude smaller than the Perl community. If you look at the number of Python people at this conference vs. the whole thing, that's probably an order of magnitude plus a factor of two maybe. So we need to grow.

Phil: The limiting factor in any magazine is advertising. When you buy a subscription, that pays for the postal service to get the magazine to you. It's advertising revenue that pays for producing the magazine. I feel like Python is in a place where there really aren't any potential advertisers except book publishers.

Guido: There aren't many people who have Python products. I mean, there are a lot of people who have products that use Python in some corner of the product. There are also lots of people who use Python but don't sell Python stuff directly. I don't know if you know it, but Industrial Light and Magic uses Python. Yahoo! bought a small company called 411, which had one of the earlier more successful web-mail applications that, not incidentally, was written in Python. Actually, I think that has happened at least twice now. I know two almost-identical cases where a small, start-up company used Python very quickly to prototype and release a product that is first of its kind. They were the first to market it, and they were so successful they were bought by a large organization, looking for a product in that area, whose efforts haven't paid off just yet. Once it gets absorbed into that big company, what usually happens is it gets rewritten in C++. That's what's happening with Yahoo! Mail, mainly because they have, after maybe two years of experience, settled on the course of the features. Now they are interested in dealing with the 80,000 subscribers they get every week or every day or so. They buy new hardware too, but you need to use that new hardware efficiently, so they rewrite the stuff in C++. If anyone had started writing that stuff from scratch in C++, he would never have gotten the product out!

Phil: They'd still be writing it!

Guido: Exactly! And because during the initial three months or so of figuring out exactly what you want, you rewrite the thing so many times and that's where you really need the very high-level dynamic range.

Phil: How common is that, that people are prototyping in Python and rewriting it in C++ or some other semi-reasonable language?

Guido: I'd say it's common for the things that become a blatant success. Lots of places end up not bothering doing the rewrite in C++, even though they always have that as part of their strategy, because the prototype works well, and performance isn't the problem in many cases.

Phil: Certainly over the years, we have addressed performance by making cheaper hardware that's faster.

Guido: Such a lovely development!

Phil: I remember UCSD P system, and other than Pascal, it isn't exactly the savior of world language. If UCSD P system had appeared with the speed of today's hardware, it would have been an amazing success because nobody would have cared. It would have solved the problem.

Guido: I manage to follow only a very small piece of the general free software world. But I am often amazed to see people post a small C program—maybe a thousand lines—that they spent a lot of time on and that does a job well. And I think, “My God! Why did they bother writing it in C? What's the performance need here?” If this is something to manage your calendar, for example, why would you bother writing that in C?

Phil: Well, that's all I have to pick on you about. Thanks for taking the time to talk to me.

Archive Index Issue Table of Contents

Advanced search

Free Clues from Eric

Doc Searls

Issue #68, December 1999

Doc talks to Eric Raymond about what he has been up to lately.

In mid-October, Eric S. Raymond's book *The Cathedral and the Bazaar* came out from O'Reilly & Associates. It contains updated versions not only of Eric's landmark title essay, but also both companion pieces, “Homesteading the Noösphere” and “The Magic Cauldron”.

We thought it would be a good idea to interview Eric about the new material in his book—and, since this historian of the hacker tribe never stops thinking, to glean a few of the ideas which have evolved out of his brain since finishing it. This conversation took place on September 22, 1999.

Doc: So now that we have “The Magic Cauldron” in final form, what new ideas are you working on?

Eric: A couple. I'm further developing the critique of conventional software-development management that I outline in the new last section of “The Cathedral and the Bazaar”. Also, I'm noticing some interesting developments in the business ecology of Linux.

Doc: Let's take the development management idea first.

Eric: I've been developing my thoughts about play being the most efficient form of work. Let's start by asking a basic question: What are the circumstances that make a programmer productive?

A programmer is productive when he is neither under-challenged nor over-challenged—neither bored and underutilized at one end of the scale, nor burdened with artificial deadlines and technical and procedural hassles at the other end. When the programmer's skill and energy are well-matched to the task, that's when you'll see the maximum productivity. The interesting insight is that these are also exactly the circumstances under which the programmer will be happiest, when his enjoyment in writing good code is maximized.

So in order to get the maximum economic leverage out of the guys you hire, you need to reduce friction costs at the top end, because you'd rather pay for their code than their grief. On the other hand, you don't want them to be bored—because when they're bored, you're not extracting maximum value from their ability.

The general prediction is this: enjoyment predicts efficiency—and not just in some fuzzy New-Agey way, but in a hard-nosed economic sense.

If you use your developers the most efficient way you can in creative work, they will be happy. On the other hand, if you have a development management style or an institutional tradition under which your developers are bored, frustrated and angry all the time, you are failing. And not just humanistically, but economically. You are failing to extract maximum productive value out of your people.

Doc: You have Dilbert's work environment.

Eric: Right. If you look at traditional software development management, you'll find that of the five classic tasks of the manager, maybe four just don't apply at all in an Internet environment. It's not so much that the traditional assumptions are wrong in some deep sense, but they're irrelevant: they proceed from assumptions that no longer match the problem. Because the assumptions don't match the problem, we've got an impedance mismatch: a lot of bored, frustrated, angry and unproductive programmers out there.

Now there is a popular “burger-flipper” school of management that says this kind of unhappiness in production workers doesn't matter as long as they're still cranking out a minimally acceptable product on the right deadlines. In fact, some managers just sort of gut-level assume that people aren't working hard

enough unless they're stressed out, so they figure the right thing to do is apply the whip every once in a while just on general principles. This is the thinking behind artificial deadlines—it's the Simon Legree theory of motivation.

But a funny thing happened on the way to the plantation—the overseers themselves are revolting. Managers are increasingly frustrated, because they're doing all the supposedly “right” things, and their jobs and their results still suck. The days that only engineers griped about this stuff are long past. Today, Dilbert cartoons hang in executives' cubicles.

Think about that. If the system was working, would the managers display contempt for it?

Doc: Tom Peters has observed that the best-selling business books are his and Scott Adams'. And he laments that all those Dilbert books are relentlessly cynical.

Eric: I'm totally with Peters on that one. When I speak to CEOs and investment bankers, I try to wake them up to the fact that the popularity of the Adams books all the way up the management chain is actually not funny. It means the system is broken. If we're going to fix it, if we're going to get our productivity to where it ought to be, we've got to start thinking humanistically about what makes people happy at work. We've got to banish the Dilbert syndrome.

Doc: What about this ecology thing?

Eric: What I'm noticing lately is that Red Hat has developed a couple of satellite distributions that are significant in their own right. One of these is Kevin Fenzi's Red Hat Über Distribution (KRUD)—which I am now running on my own machine, by the way. The other is Mandrake.

Doc: These are cases where others are leveraging Red Hat.

Eric: Right. Both looked at the Red Hat distribution and said, “I see a significant value-add here that Red Hat is either not interested in or not in a position to exploit.”

In the case of KRUD, Red Hat doesn't put out updates often enough. So what Kevin does is press CD-ROMs every month on a subscription basis. A new one shows up in the mail, you stick it in your drive, go through the update procedure that pulls all the new RPMs off the CD-ROM and installs them, and you are completely up to date.

I think of this as the magazine model of operating system publishing. Want all the latest updates? Subscribe—it's just $36 a year. In return, you don't have to spend a lot of time downloading stuff and worrying if you're current.

Doc: Who's the customer?

Eric: KRUD is particularly valuable if you are a site with serious security requirements. Want to be sure you have all the latest security patches? That's what Kevin does for you. He's the maintainer of the Linux Security HOWTO. He's at http://www.tummy.com/, by the way.

Doc: Why can't Red Hat do this?

Eric: Scale, I think. They have to ship CD-ROMs in sufficiently large runs to carry their other expenses. They would be bleeding all over the place in fulfillment and marketing expenses if they tried to compete in this new space by going to a one-month rather than a six-month release cycle.

Doc: Red Hat only scales higher up the value chain.

Eric: Right. Kevin can do this because it's a relatively small effort for him, and he doesn't carry all of Red Hat's development and marketing overhead.

Doc: What about Mandrake?

Eric: Their value-add is that most people have Pentiums now, and Mandrake can take all the source RPMs and recompile them with Pentium optimization so your machine runs 30% faster. Again, it would be too expensive for Red Hat to have two distributions, one optimized and one not.

Doc: Does Red Hat benefit here?

Eric: Sure. Red Hat is certainly growing their support market this way. If I'm a Fortune 500 site and I have a subscription to KRUD and have a support requirement, do I go back to Kevin? No, because I know he's just one guy sitting in a room in Colorado somewhere. So I go back and buy a support contract from Red Hat.

Doc: And if you're in the same position with Mandrake?

Eric: You may go to LinuxCare. I believe they're Mandrake's partner for service. So the Mandrake/LinuxCare combination is competing with Red Hat a little more directly for service revenue. Still, the Mandrake guys have a good working relationship with the Red Hat guys. In effect, Mandrake is acting as an R&D arm

for Red Hat, because the changes they make, to fix and clean up things, can get folded back into future Red Hat distributions.

Doc: And the ecology point?

Eric: Because everybody is playing by the open-source rules rather than competition, there's a potential for cooperation to make more money. There is a potential for single distributions to speciate into mini-ecologies of mutually supporting distribution variants. These guys all share core DNA, and that's what makes the ecology work.

Doc: I imagine there is some analog in original markets.

Eric: Oh, sure, it's a value network in Christensen's sense. [Clayton Christensen is the author of *The Innovator's Dilemma*, an award-winning book on disruptive change in technology markets—Editor]

Doc: Have you got any more books in the pipeline?

Eric: Yes. I'm about halfway through writing the first draft of *The Art of UNIX Programming*, a book on how to think like a UNIX guru. It's aimed at a lot of the younger Linux programmers who have picked up bits and pieces of the UNIX design tradition, but don't really have the whole Zen. This is not their fault—because like Zen, it's only been passed along from master to student up to now—"a special transmission, outside the scriptures". It's time to write it down.

Doc: Any other significant developments for you?

Eric: I'm doing most of my programming in Python these days. I've moved away from C.

Doc: Python is hot. It's taking our own people by storm.

Eric: Under present economic conditions, using a language in which manual memory management is required almost never makes sense. The thing about C is that you have to manage your buffers and dynamic storage by yourself. You don't have a garbage-collecting interpreter doing it for you. It's a huge source of problems—classically, something like nine out of ten of the runtime errors in C programs are memory-management screwups.

It used to be, back when machines were more expensive and programmer time less so, that it was worth hand-tuning resource utilization and putting in all the debugging time necessary to compensate for the high error rate. But unless you're doing systems programming or really heavy-duty number crunching,

that tradeoff just doesn't make sense any more—not with cycles and memory as cheap as they are now.

Doc: Is that Python's main virtue for you?

Eric: One of several, but in my opinion it is the single most important one.

Doc: But other languages do the same thing.

Eric: Yes, there are lots of interpretive languages that do garbage collection, but very few are as well-designed as Python. The alternatives don't scale well; they make it harder to read and modify large volumes of code. I used to write a lot of Perl, until it got too painful. And the less said about Tcl, the better.

Doc: Closing thoughts?

Eric: Just a sound-bite version of what's beneath most of the open-source business models we're seeing out there. “Linux is free. Clues are not.”

Archive Index Issue Table of Contents

Advanced search

Product Review: Diffpack

Jim Moore

Issue #68, December 1999

Diffpack is an object-oriented toolkit for creating numerical analysis applications. It provides high-level building blocks which may be put together to rapidly create a high-quality application for solving partial differential equations.

Diffpack

- Manufacturer: Numerical Objects AS
- E-mail: sales@nobjects.com
- URL: http://www.nobjects.com/
- Price: Varies according to license (see below)
- Reviewer: Jim Moore

Diffpack is an object-oriented toolkit for creating numerical analysis applications. It provides high-level building blocks which may be put together to rapidly create a high-quality application for solving partial differential equations. The software package has an accompanying book, *Computational*

Partial Differential Equations (CPDE) by Hans Petter Langtangen (Springer-Verlag).

I must admit I was immediately impressed with the book because it was typeset with TeX by the author, and as any TeX user will tell you, using it is a sign of intelligence. The book is well made and beautifully set. It is written in a style very similar to *The Visualization Toolkit* by William Schroeder, et al. (Prentice Hall Computer Books) in which a careful description of the approach and methodology for building numerical algorithms for solving partial differential equations (PDEs) is given, with all the examples being demonstrations of the Diffpack software.

CPDE begins with a strong and well-supported endorsement of object-oriented programming. It then proceeds to describe PDEs of increasing complexity and numerical approaches which can deal with them. The book explains the difficulties of the mathematics as well as the intricacies involved when the PDEs are “linearized” into systems of algebraic equations and solved in various ways. Techniques for performance optimization are also covered. With each level of complexity, relevant sample problems are solved using the Diffpack software to demonstrate how the problem can be solved. *CPDE* focuses more on the finite element method than the finite difference method, probably due to the author's experience, but gives sufficient coverage to both.

The main chapter topics accurately describe the content of the book:

- 1. Getting Started
- 2. Introduction to Finite Element Discretization
- 3. Programming of Finite Element Solvers
- 4. Nonlinear Problems
- 5. Solid Mechanics Applications
- 6. Fluid Mechanics Applications
- 7. Coupled Problems
- Appendix A. Mathematical Topics
- Appendix B. Diffpack Topics
- Appendix C. Iterative Methods for Sparse Linear Systems
- Appendix D. Software Tools for Solving Linear Systems

There are 127 exercises to help a student of numerical methods deepen her understanding of the topic. The demonstrations make wide use of tools commonly available on Linux systems such as Gnuplot, Plotmtv, Matlab, Vtk and Xmgr. The scientific Linux user will feel very much “at home” reading this text. My usual complaint for technical books is that they either go too far with

examples and don't provide enough background, or they do the opposite and go too far with theory and leave the reader with no concrete way to apply it. In my opinion, this book has struck the balance well. I wholeheartedly recommend it as a general text on the topic. If you plan to use Diffpack, it is a requirement.

Platforms

Diffpack is available for all major UNIX flavors and the Win32 platform. I tested the software only on Linux. There are four types of licenses: Commercial Developer, Non-Profit Developer, University Developer and University Classroom. They initially cost $9995 US, $3150 US, $995 US and $1995 US, respectively. The classroom license allows five concurrent users. Annual service contracts cost roughly 13.5% per license and additional licenses cost less than the initial license. There is an additional fee for multi-license, multi-platform support as well. For further price information, contact Numerical Objects AS directly.

Though I did not test it, a plug-in called the Adaptivity Toolbox is available which enables any application to implement adaptive grid technology. It comes at an additional price ranging from $995 US to $3150 US depending on the type of license. This tool is essential for some applications and should be added directly as part of the cost of the purchase price. If your problem involves changes of scale of an order of magnitude or more, you will probably need this tool.

Installation

I installed the software on my Red Hat 6.0 Dual Pentium-II system from the demo CD-ROM, which may be ordered directly from Numerical Objects at http://www.nobjects.com/. The instructions were complete and the installation process was as simple as running a script, setting two environment variables and making a one-character change in a Makefile to reflect the name of the compiler on my system. However, if you happen to have a version of **egcs** newer than 1.0.3 and no longer have 1.0.3 lying around, I have two words for you—*forget it*. Diffpack *must* have egcs 1.0.3, and if you have a newer version, there is no way to compile applications with it. I ended up NFS-mounting the diffpack distribution on an older computer I hadn't gotten around to upgrading, so that I could get the examples to compile. This is not Diffpack's fault, but a show-stopper nevertheless.

The CD contains over 50 demonstration applications which directly correlate with the book. The CD-ROM has very well-structured HTML documentation which explains each demo and extracts them from the CD via a link. Each example is structured exactly alike, with source code, a Makefile and a README

that explain how to build and run the example. All examples have a “Verify” directory which contains input files for verification simulations. The examples are usually designed with some sort of progression in mind, walking the user through the various capabilities of Diffpack and the helper applications which accompany it. Some applications require the licensed version of the code to run, but the CD provides pre-built executables for some of these. You will also need Perl-Tk to run any of the graphical interfaces to diffpack applications. My Perl-Tk is not current enough, so I didn't have the pleasure.

After working through the installation and application compilation process, it is clear that Diffpack is exceptionally well-designed. Quite a bit of thought has gone into making the installation process work well on any *supported* platform. The software installs simply, but still gives the user complete control over how the installation is done. Want to change (supported) compilers? No problem. Want to use VTK for visualization? No problem. It takes a little reading to make these things happen, but it is usually as simple as altering an environmental variable or setting an option argument for **make**.

The Good

- Excellent book
- Powerful software
- Well-written
- Well-documented

The Bad

- Extremely particular about libraries and egcs versions
- Steep learning curve

Usage

Diffpack is like a fractal pattern viewed from a distance: looks simple and is simple, and the closer you look, the more there is to see. Again, unless you are running Red Hat 5.2 or a similarly time-stamped Linux distribution, your efforts will most likely fail. The applications may compile, but will dump core because you have an older f2c library, for example. Though my efforts failed, Numerical Objects reported to me that users of the Red Hat 6.0 distribution have successfully run Diffpack, so don't lose heart if you already made the change. It will just involve a little more work. Once you have the correct installation base —watch out!

I compiled and and ran examples from each of the sample directories: fem, fdm, linalg and app. The examples cover everything from “Hello World” to the

wave equation to coupled heat and momentum transfer. The first, Heat2, from the fem series, gave me chill bumps. It was a simple 2-D transient Fourier heat conduction problem. I compiled the application (**make MODE=opt**), ran it (./test4.sh), ran LaTeX on one of the output files, and saw a complete report of the simulation: input conditions, routines called, solution history and graphical results. The report was also generated in text and HTML format. The input file was about 20 lines long and set up the grid (rectilinear), boundary conditions and run controls. The shell script which managed the application contained two lines. The application code was only 350 lines of C++ and most of that was for managing a text menu system built into the application. A bare-bones application of this type would probably have required about 30 lines of code.

The true advantage shows up when you want to change something, like solvers. Change one line, recompile, and you are done. This package effectively removes the tedium from building numerical solvers and allows the numerical scientist to deal with the problem at hand: getting a solution or proving there isn't one.

Some time will have be spent with the documentation to understand how to build applications, but once done, I suspect there is little you cannot do with this package in the field of numerical modeling. The package follows the UNIX paradigm of making many small utilities which may be combined to create the result you want. For example, results are presentable in so many formats because of helper applications such as **simres2gnuplot** and several other simres2* applications. Freedom and extensibility appear to reign supreme.

Of course, with freedom comes responsibility and a learning curve. Diffpack provides C++ classes which may be used to build applications. The classes are very high level, so you can call things like “GridFE” or “ConjGradNonLin” to deal with finite element grids or a conjugate gradient solver, respectively. The example programs are a very instructive aid to developers who want to build applications with Diffpack. I would liken the experience to learning to use VTK. However, because I am a numerical modeler and not knowledgeable in C++ matters, I can't report on the difficulty of building an application. I can report that pre-built applications are relatively short, easy to read and understand. With the book to aid you, it should not be too hard to become proficient if you already understand numerical methods. Because the package was built with an object model, fine control of the high-level classes is possible through sub-classes.

As with most software, you don't realize its shortcomings until you are very heavily invested in it. I don't know if it is possible to exercise fine control on mesh topologies during a simulation or vary convergence criteria by node location, for example. Most numerical simulations packages invoke heuristic

“tricks” to get the job done on difficult problems. The diffpack implementation appears to be quite pure, and it may be difficult to get away with some tricks that practitioners of the “art” are familiar with. On the other hand, because it is pure, it may be the first opportunity to solve many of the problems engineers have cheated on for years. Interface tracking comes to mind.

Diffpack appears to be a well-written, well-documented tool which does exactly what its own press suggests: “Closing the Gap”. Perhaps “filling” is a more appropriate term, because this stuff is as hard as it ever was, and the “gap” between understanding a physical phenomenon and finding a reliable computational solution is large. This tool makes the gap easier to span.

Jim Moore and his wife Kim simultaneously work to transform two little urchin daughters into respectable and productive human beings and be productive and respectable themselves. They are coming to believe the two goals are mutually exclusive. Jim is developing distributed, object-based, numerical software for his startup, URS Technologies, LLC in Columbus, OH. He has a BSME from UT Austin, and a Ph.D in metallurgical engineering from Ohio State. He may be reached at jmoore@qn.net.

Archive Index Issue Table of Contents

Advanced search

Castlewood ORB

Patrick Lambert

Issue #68, December 1999

The ORB is both inexpensive and contains a good amount of data (2.2GB per disk, which can compress up to 6GB).

- Manufacturer: Castlewood Systems Inc.
- E-mail: castlewood@castlewood.com
- URL: http://www.castlewood.com/
- Price: $199 US for drive $29.95 US per disk
- Reviewer: Patrick Lambert

In the past few years, I have bought a lot of hard drives for various reasons. From testing to storing Internet downloads and archiving, I always needed more disk space. When I began looking at the available removable media devices, I wasn't impressed. Most contained only a few hundred megabytes and were very expensive. Then I came across the ORB. The ORB is both inexpensive and contains a good amount of data (2.2GB per disk, which can compress up to 6GB). A month later, I was buying a parallel port ORB device and two disks.

Device Information

The Castlewood ORB is a device capable of supporting removable disks containing 2.2GB of data. The ORB comes in several versions:

- Internal IDE
- Internal SCSI

- External SCSI
- External Parallel

The ORB is advertised as a good way to store important data, have portable information, store digital images and other media, back up a hard disk and store Internet downloads. The manual specifies that a Pentium 100 or better system is required.

Hardware

I bought the ORB from one of the on-line resellers and received it in less than a week. The ORB box is quite large and contains the ORB drive, a disk full of utilities, an interface cable, a power supply, a user manual and an installation floppy. The drive is nice and looks modern on a desk. It is black with a small door for the disk on the front. The ORB power supply is like any other power supply, appearing very common. The interface cable is very short (about two feet), and you cannot use a normal printer cable to link the drive to the PC. The drive has an output port to connect a printer. The manual recommends that no other parallel device be used with the ORB and the printer.

One thing I found out is that even if they include a removable disk in the package, you actually need to buy another one since the included disk has 1.6GB of tools on it.

Included Software

The installation floppy contains drivers for various systems, including Windows 98, Windows 3.*x* and DOS. The included media has the following Windows tools:

- ORB Tools: a set of tools and utilities used to manage the drive, eject a disk, scan the drive and other similar functions.
- 1-Click Backup: a full system backup can be done with this utility by right clicking on the ORB's drive letter.
- Advanced Backup: this full suite of backup tools is for professional users.
- Duplicator: this tool allows you to duplicate multiple ORB disks.
- tracker: tracks your files on all your disks.
- Rescue: this small utility lets you restore your boot drive in emergency situations.

The disks come formatted in FAT16, which is readable and writable by Linux, DOS and Windows. They can be partitioned and re-formatted like normal hard disks.

Windows Installation

The installation in Windows was very easy. When inserting the driver's floppy, you can install the drive in less than 30 seconds. The ORB device will then appear as a drive letter, seen from Windows as a removable hard disk. Installing the tools provides all kinds of interesting functions, including backups and a software-based eject button.

One thing I noticed is that when the RealPlayer is running and I insert a disk in the drive, then try to access it from Windows Explorer, it hangs the system. This may be a bug in the ORB driver, the RealPlayer or both.

The Good

- Inexpensive
- Easy to use
- Large storage size
- Lots of software
- Works with Linux, OS/2, DOS and Windows

The Bad

- Unreponsive when writing large files
- Much slower than an IDE drive

Linux Installation

The Linux installation was also easy. The ORB device acts like an OnSpec device, which is supported by Linux. Here is a quick installation guide, followed by the steps.

```
/sbin/modprobe paride
/sbin/modprobe on26
/sbin/modprobe pd
mknod /dev/pda5 b 45 5
mount -tmsdos /dev/pda5 /mnt/orb
```

The first step is to load the necessary modules. Three modules are required, and unless you link them in your kernel, you need to load them when you boot up. The first module is **paride**, the parallel driver which handles IDE devices. The second is the OnSpec26 driver, which should find the drive. The **pd** module should load the disk.

Once the drive and disk are found, you need to mount the disk. By default, the media is formatted as an extended FAT16 partition. The first four partitions in a Linux file system are “primary”; number five and up are called “extended”. This means you need to create a device named pda5. The **mknod** command will do

just that. The **mount** command will mount the partition in /mnt/orb, assuming you have created that directory.

If you have problems mounting the drive, you may want to look in your syslog files. These should contain report messages from the module's loading and tell you what is wrong.

Performance

The first time I used it, I noticed how silent it was. You can't even hear it write to the disk. When you first insert a disk, you hear the same sound as when you boot a system, and the BIOS loads the hard drive. When the disk is loaded, you can mount it and read/write directly to the disk.

I noticed two problems when working with the ORB. First, when the system has large files to write to a removable ORB media, it becomes very occupied and unresponsive during the time it is writing to the disk. I am assuming this is because it has to write via the parallel port, and the system needs to send the data at a fixed speed and compression.

The other thing I noticed is the speed, which while better than every other removable media I have tried in the past, is still not as good as an internal IDE drive. The 2MB/sec advertised is the burst speed. I found the write speed to be around 100 to 200KB/sec, transmitting around 10MB in a minute.

Conclusion

With its low cost, ease of use and included software, the ORB is a good product to buy. Now I use it to do all of my local backups and store important archive files which I may need in the future, such as Netscape Communicator and Word Perfect 8.

I think the Castlewood ORB is the best removable media yet, and it is great that it works in most popular operating systems including Windows, OS/2, DOS and Linux. I would like to see Castlewood provide formal support for Linux and their web site advertise the fact that ORB works on Linux.

Patrick Lambert is currently a student in Computer Science at the University of Montréal. He has been using various UNIX and Linux systems for five years,

doing software development and systems administration. He can be reached at drow@darkelf.net.

Archive Index Issue Table of Contents

Advanced search

MailStudio 2000

Jason Kroll

Issue #68, December 1999

MailStudio 2000 is a web-based e-mail server from 3R Soft for Linux, Solaris, HP and DEC that provides web-based e-mail service for users.

- Manufacturer: 3R Soft, Inc.
- E-mail: sales@3rsoft.com
- URL: http://www.3rsoft.com/
- Price: starts at $299 US for 30 users
- Reviewer: Jason Kroll

E-mail changed the way we communicate, and now the Web has changed the way we e-mail. Web-based e-mail has numerous advantages over traditional e-mail: access from any web terminal, no need for e-mail client software (such as Pine, Elm or Mutt) and multi-platform support (users have the same interface on Linux as on any other platform). MailStudio 2000 is a web-based e-mail server from 3R Soft for Linux, Solaris, HP and DEC that provides web-based e-mail service for users. It is accessible through web browsers such as Netscape and also through POP3-based mail programs such as Eudora. It can even be implemented on top of existing POP3 servers to provide a local interface. Unlike many web-based e-mail servers, MailStudio 2000 apparently aims to be a quality, functionality-oriented e-mail utility, rather than a commercial excuse for running advertisements (though you can, if you must, put ads on MailStudio

2000). Three main elements of a web-based e-mail server are its interface to the users, its interface to the administrator and the technical features of the server.

The User Interface

The uniform user interface offers a number of traditional e-mail features, such as mailboxes and address books, as well as some things which are quite modern (or even trendy), such as the ability to send e-mail as HTML with italics and graphics and all of that bother. (This tends to annoy people, actually.) By default, MailStudio keeps Inbox, Sent and Trash folders, and you can apparently add as many as you like, which will be kept in the user's folder menu. The Options folder contains entries for changing your preferences, folders, external mail configuration (for POP3 servers), address book, signature, user information and password. There is also a Question & Answer board for the system and a User Search function for finding others. The last option is “Log off” which kills the encrypted cookies used for user authentication.

The Good

- fast
- extremely easy
- flexible
- it works

The Bad

- a bit pricy
- somewhat insecure
- flickers

In order to use MailStudio 2000, a user must have Java and cookies enabled. In addition, it seems to take a while to load, with a lot of flickering during initial contact. The software is constantly being developed, so maybe this flickering will go away soon. Everything else is simple, like any other mailer. People who don't like command lines, **vi**, or typing in general will like this entirely graphical user interface. MailStudio 2000 used to have multi-layer folders and address books, which I think is a neat idea, but they eliminated multi-layers because they weren't entirely cooperative with frames.

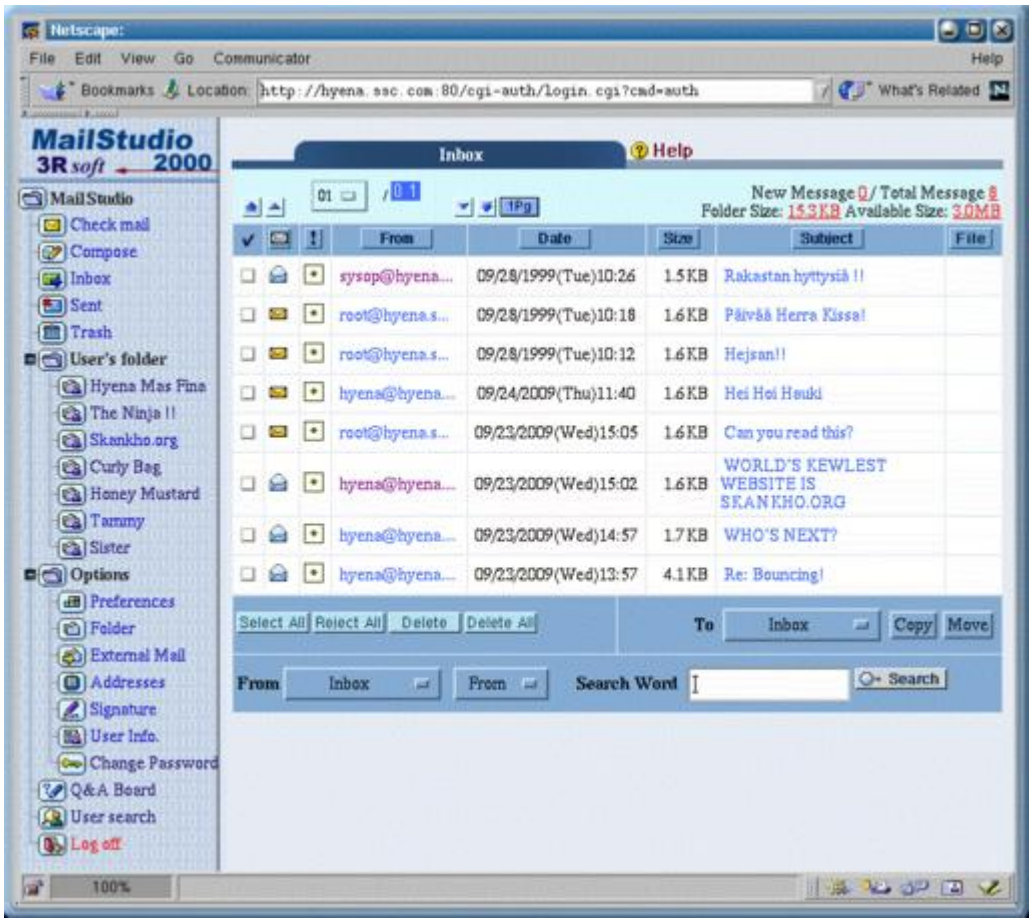


Figure 1. A Picture of the User Interface

The Administrator Interface

From an administrator's viewpoint, MailStudio 2000 has a number of good qualities. Installation is easy, and should take no more than a few minutes. Likewise, starting and stopping the server is quite simple with **start** and **stop**. Administrators can change the user interface just by changing the HTML files, making all sorts of customization possible, as well as advertisements if you really want them. (Well, it wouldn't be the Web without ads.) The client/server model of MailStudio is based on open standards, so it integrates easily with other software. As an example, user database and mail files can be automatically backed up by **cron**; the manual gives the cron entries for doing this. For normal management of these user database and mail files, MailStudio's intended interface is a web based menu which is accessed by logging in as sysop. The interface provides menus for user management, application, admission and admin's (system) preferences. Of these, the administrator's preferences menu has the important, system-wide, technical parameters.

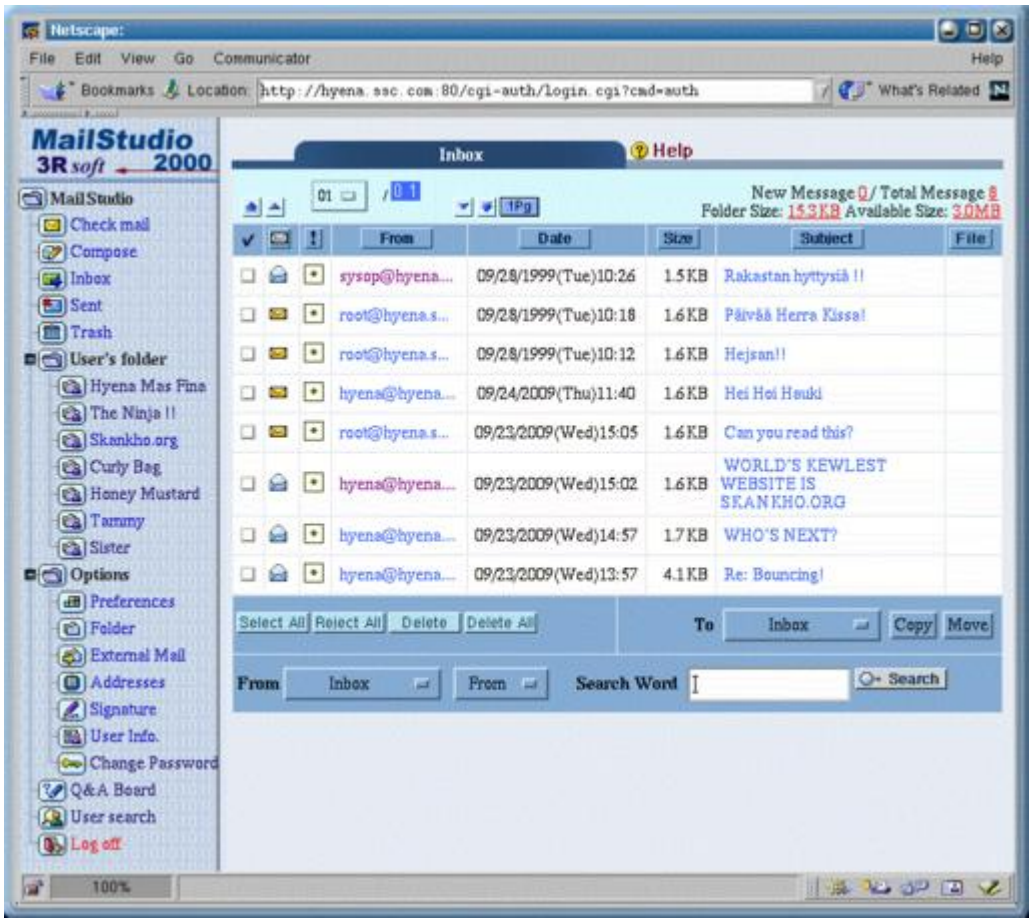


Figure 2. Administrative Interface

Technical Features

The server has a number of functions and features to help it deal with the complicated issues which arise in something seemingly as simple as web-mail. Not all mail is processed following the same protocol, so in addition to the standard Sendmail and SMTP, MailStudio is also able to support POP servers and retrieve mail from them.

Security, such as it is, is achieved by way of cookies, which are encrypted with MD5, and customers can order SSL services as well. The server supports numerous standards such as MIME (Multipurpose Internet Mail Extensions), multipurpose and dynamic HTML, ISO2022, Quoted Printable, BASE 64, uuencode, two-byte character sets and multiple-language environments. Embedded in the system is the CodeBase 6.3 database for dealing with all the data.

Overall, the server is flexible, adaptable and generally cooperative. For example, you can assign MailStudio to a specific port in order to avoid conflicts with a web site you are already running, and you can implement virtual mail hosts to achieve multi-domain support. In terms of resource consumption, MailStudio allegedly can support over one million users and can deliver fast

service even during heavy loads. (I could not exactly verify this point, so we'll have to take their word for it.) The system is self-contained, with safeguards, simple methods and standard error procedures, and requires very little maintenance.

Security

Holistically speaking, the problem with networks is security. Every packet of information usually travels through several machines, and all it takes is a packet sniffer at any point along the way and your data can be intercepted. This is a reason not to send unencrypted credit-card information over the Internet. (This is also why we want the U.S. government to allow stronger encryption.) However, when it comes to dealing with sending passwords over the Internet, it's a complicated problem to fix. For this reason, web-based e-mail has always been vulnerable to security breaches, whether through accessing a page directly to bypass login or through packet sniffers routing out people's passwords. Cookies have largely put an end to the former problem, but the latter is still with us. Fortunately, crackers (as Eric Raymond likes to call them) don't have too much interest in reading other people's web-based e-mail. Still, at login, your user name and password are sent unencrypted across the network to the MailStudio server, as are the e-mail messages you send and receive. Although this is the same situation as with TELNET or almost anything else, it's not safe and something must be done (hence secure shells). 3R says it can purchase and implement an SSL package from a third party and integrate it into MailStudio in order to have encrypted first-pass user name and password data. I think 3R should just write SSL into MailStudio since it is, after all, an open protocol. Maybe it'll show up in a later version; I certainly hope so. If systems were secure in the first place, fewer curious people would be in prison, and script kiddies wouldn't seem so annoying.

MailStudio of the Future

Even though it seems like a simple task, implementing a web-based e-mail server is complicated due to the numerous protocols, standards and platforms. Because of this, 3R concentrated on performing one task simply and doing it well. Much effort was put into making this software run easily, and I think it is a success. I'm not a web type (three cheers for Lynx), and I managed to have it up and running in less than ten minutes.

One strange quality of MailStudio 2000 is that the advertisement 3R sent with it didn't exactly correspond with the features on the server. The reason? MailStudio is still being improved. Now that the base product exists with everything working, 3R can concentrate on improving the features, such as search routines or user management. I have been told by 3R that their later releases will have all sorts of neat things. An on-line demo at http://

www.mailstudio.com/ shows the system and demonstrates the current features.

I was definitely impressed with MailStudio 2000's ease of use. While I am concerned about the insecure first pass of user name and password, and the flickering during the first screen load, it seems to be quite resource-minimal. One strange thing I noticed is that all e-mail I receive from MailStudio comes with paragraphs formatted as single lines, which is a bit weird. Still, with web-based products so easy to use, it's not hard to understand how the Web has become so immensely popular and populated. MailStudio is more expensive than I'd like (which means it costs more than Apache), but I suppose if you have the resources to run a web site, perhaps you can afford the price—still, ouch.

MailStudio 2000 is targeted not only at free e-mail providers but at educational and commercial organizations, Internet service providers and government and public institutions. It received five Golden Penguins from TUCOWS and is listed as a Sun Microsystems software solution. You can download a trial version (which supports five users or so) from http://www.3rsoft.com/. It is Y2K compliant (well, wouldn't it be fun to have a product named MailStudio 2000 that wasn't Y2K compatible?) and it just so happens I'm running it successfully from a VArStation in which the BIOS clock thinks it's the year 2036 (though server problems will arise in 2038). If you're looking for a web-based e-mail server for your Linux box, definitely take a look at MailStudio. It's already very neat, and I'm told the next version will be a total web-based e-mail solution.

Jason Kroll (info@linuxjournal.com) came to Linux from the world of Commodore 64 and Amiga where he grew up—he likes GNU/Linux so well he's not going back. He has been spotted on the streets of Silicon Seattle handing out Linux CDs to passers-by.

Archive Index Issue Table of Contents

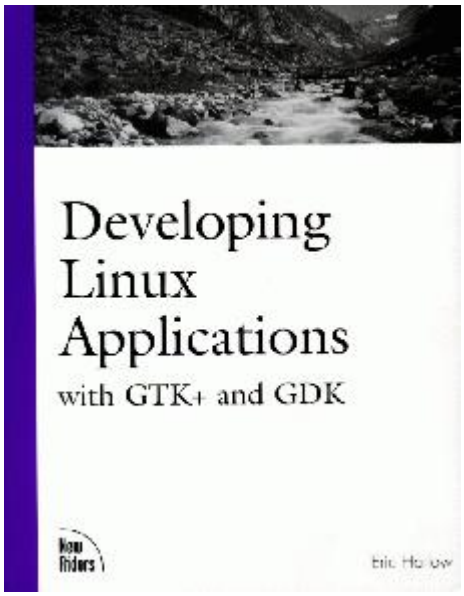
Advanced search

Developing Linux Applications with GTK+ and GDK

Michael Hammel

Issue #68, December 1999

Although not meant for the experienced user, this book does have its high points, even for the old hands of GTK+ like me.



- Author: Eric Harlow
- Publisher: New Riders
- URL: http://www.newriders.com/
- Price: $27.99 US
- ISBN: 0-7357-0021-4
- Reviewer: Michael J. Hammel

I've been working with GTK+ since its early introduction with the GIMP several years ago. I've watched it mature and become a fairly sophisticated widget set—one of the best I've seen for use under X. Until recently, GTK+ could be mastered only by those willing to dig into the source code, a feat not always for the faint of eyes and fingers.

Fortunately, the first of what I'm sure will be many texts on GTK+ has already hit the shelves. Eric Harlow's text *Developing Linux Applications with GTK+ and GDK*, published by New Riders, is an introduction to the world of GTK+ that many refugees from the world of MS Windows will find useful. Although not meant for the experienced user, this book does have its high points, even for the old hands of GTK+ like me.

The book is laid out in four sections:

1. Programming in GTK+
2. Application Examples
3. Drawing, Color and GDK
4. Extending GTK+

Chapter 1 isn't actually a chapter—it's the preface. The rest of the book starts with GTK+ basics and moves on to more complex issues with each section, which is made up of two to six chapters. The layout is a little strange with respect to example code. The chapters in the first section are introductory in nature, covering glib and some basic GTK+ features. The chapters in the second section each cover an example application, aimed at showing off what was learned in the first section. The third section, however, has its example code as the last two of its four chapters. The fourth section has the examples intermixed in each chapter. You might not really notice this layout pattern unless you step back and look at it from a reviewer's point of view, as I did, but it is a little strange to see the material broken down in that way. I would have expected examples simply to be intermixed with each chapter.

Chapter 2 describes glib, the library of utility routines for handling hashes, linked lists and memory management. I don't personally use glib all that much —I have simple library routines of my own to do these things I'd written long ago and carry from place to place. However, some widgets make use of certain glib features directly, so it helps to become familiar with what it is and what it does.

The next chapter shows how a simple GTK+ application is laid out. This leads into Chapter 4's introduction to basic widgets: buttons, labels, menus and text-entry widgets. Chapters 5 and 6 also describe widgets. Although several chapters are on widgets, you need to realize they can be fairly complex and offer many options and their accompanying support and convenience routines. No widget description takes up more than two pages (except the Clist and Text widgets, which are introduced very late in the book). Don't expect detailed API descriptions here—this text is very introductory in nature.

On pages 27 and 28, Eric did get one minor thing wrong—he said GTK+ came from the GIMP, and that it stands for the GNU Image Manipulation Project. The P in GIMP actually stands for “Program”. He also said GTK+ was born over licensing issues relating to the GIMP's original widget set, Motif (though not in so many words). I don't believe that was the driving reason. Motif was limited in some areas, and not having source made it difficult to extend it. Plus, I think the programmer wanted to work on a widget set from scratch anyway.

Eric describes accurately how to build applications using a single command line, including the recommended use of the **gtk-config** script. One additional item might be the use of the **GTK_CONFIG** environment variable. If you use this in your projects, you allow people who have multiple versions of GTK+ installed (it's a constantly evolving library) to build whichever version of the library they need.

The section on container widgets in Chapter 3 could have used some finer detail. Packing with boxes and tables is often a confusing area for recent converts.

Chapter 4 starts off with some discussion on the casting of widgets. GTK+ widgets are generic when created, so to use them with type-specific functions such as **gtk_label_set_text**, you need to cast the widget to the correct type. This is done with macros, such as **GTK_LABEL**. The discussion in the text is good but brief. A table showing inheritance for widgets would have been nice, but this isn't a reference text.

On page 51, Eric had a somewhat strange segue into the paragraph on signals after finishing off the discussion on casting. The information is useful, though it seemed to come in out of the blue.

The rest of Chapter 4 introduces the reader to labels, buttons, text input (known as entry widgets), simple lists, combo boxes (lists with text input fields), option menus and generic containers (widgets that hold other widgets). This is where some discussion on naming schemes for widget-specific functions would probably have been helpful for those willing to rummage through the GTK+ source code.

Chapters 4, 5 and 6 are descriptions of various widgets. Chapter 7 begins the second section and its example applications. The use of the notepad application as an example was a good idea—it's a familiar application to developers—but the section on the text widget should be tempered with the knowledge that it's likely to change in the next major release of GTK+. Owen Taylor, the man who wrote the original text widget, has said the design doesn't lend itself well to new features, and he has some better ideas on how to

redesign it. Whether that will mean applications using the widget will no longer work is unknown. These changes will probably not happen for some time, however.

One problem with printed texts is the amount of time it takes publishers to get them to market and onto shelves. Because of this, the text is a little behind the curve for the current version of GTK+, which is version 1.2. I believe Eric was working with version 1.1.5 at the time he wrote the text. Some of the examples, although they work, use outdated functions. For example, he uses **gtk_label_set** in many places in the code. The correct function to use with GTK 1.2 is **gtk_label_set_text**. You can check the GTK+ web site (http://www.gtk.org/) to find changes from GTK 1.0 to 1.2, if you are uncertain about any of the other code in the book.

One common concern about labels, not mentioned until the last chapter on creating your own widgets, is that labels are subclassed from the GtkMisc widget, which means labels don't receive signals. Thus, they must be wrapped in other widgets. For event handling, you would drop the label inside a GtkEventBox widget. Remember this bit of information—you're bound to need it eventually.

Chapters 10 through 13 make up the third section, covering issues relating to color and drawing with low-level GDK routines. I found chapter 10 to be the most useful, as I finally figured out how to do double buffering. Double buffering is simply a matter of drawing in an off-screen (non-visible) region—a pixmap—and then copying that region over the currently visible version. The technique is simple once you learn it. It's a subtle piece of code, and often hidden in the bowels of more expansive subroutines. Eric explains double buffering clearly and provides good examples of how to use it.

Chapter 11 moves into styles, colors and fonts. The discussion here is like the rest of the text: brief but accurate. The styles section, however, talks about the use of styles programmatically. It doesn't discuss them from the point of view of gtkrc files. Styles are, specifically, an API term for GTK+, but adding *themes* (what an end user might refer to as a style) is a big deal with applications using GTK+. The gtkrc files add a “look”, but are far less sophisticated than the application-default files common with Motif. The latter allow external definition of command-line arguments, for example. Currently, gtkrc strictly handles look-and-feel issues.

There is no discussion on the use of command-line arguments with GTK+. Motif programmers are probably used to Motif's built-in command-line parser. GTK+ doesn't have such a thing. If you plan on having command-line arguments, you will probably want to use **getopt_long**, unless you're worried about portability

to non-Linux platforms. In that case, you may have to roll your own parser, since not all UNIX boxes have getopt_long.

Source code for the book is available from New Riders' web site. It comes in gzipped tar format, with the examples broken out into chapter-by-chapter directories. There is no top-level Makefile, but the individual directories build easily enough. Be sure to have the **gtk-config** script in your path. Most of the applications worked fine for me. The *molecule* example didn't seem to draw any molecules, unfortunately. And the double buffering example—one clock without and one clock with—didn't show the advantage of double buffering on my system, because it's fast enough (400MHz) that the redraws in the unbuffered version were hardly noticeable. But you *should* pay attention to that example. Double buffering is a good thing to learn if you're doing any serious amount of drawing with GDK.

Considering the introductory nature of this book, I wonder how useful the last chapter on writing your own widgets will be to others. This space might have been better used to cover more details on widget specifics.

Eric's writing style is succinct and clear. The book has few typos—on-line documentation is notorious for poor editing, which is one reason why printed, professional documentation is here to stay. Reading this book is certainly easier than reading many of the on-line documents I've seen.

This book is a beginner's guide to writing GTK+ applications. It is not a reference, and leaves much to be desired regarding in-depth widget explanations. If you need only an introduction and you're looking to migrate to Linux and GTK+, it isn't a bad place to start.

Michael J. Hammel (mjhammel@graphics-muse.org) is a graphic artist wanna-be, a writer and software developer. He wanders the planet aimlessly in search of adventure, quiet beaches and an escape from the computers that dominate his life. He is the author of The Artists' Guide to the Gimp.

Archive Index Issue Table of Contents

Advanced search

lpd: Getting the Hard Copy

Michael Hughes

Issue #68, December 1999

How to set up local and networked printing services in Linux.

In the 16th century, the printing press opened up a whole new world of communications. Print was the first mass communications medium, paving the way for books, magazines, newspapers and all the other amenities of this generation. Although we live in a world dominated by computers and electronics, the printing press still plays an important role in our everyday lives. This fact is especially apparent if you are reading the print version of this magazine.

Today, the power of the printing press is available to the individual. Computers, printers and copy machines allow virtually anyone to communicate effectively to any number of people. Better still, it's easier and cheaper than ever.

One of the most common questions asked by newcomers to the Linux world is how to get their printers working with Linux. There are, in actuality, a few different ways to accomplish this task. First, however, you must enable **lp** support in the kernel and recompile. This is done in most kernels already. To check if your kernel is ready, plug your printer in and watch your kernel startup messages. If you see references to lp0 or lp1, your kernel is configured for parallel printing.

Setting it Up

The most primitive way to print a text file is simply to use the **cat** command to send it to the printer at /dev/lp1:

```
cat filename.txt > /dev/lp1
```

This will catenate the file (in this case, filename.txt) to /dev/lp1, your printer device. Replace /dev/lp1 with the device name of your printer, if it is different.

The main problem with printing text files this way is that most people get the dreaded “staircase” effect. It makes the printed text look

```
Something
   Like
      This.
```

This is not acceptable, so most people use **lpd** (line printer daemon) to print files. If you don't have lpd already installed, it is obtainable from ftp://metalab.unc.edu/pub/Linux/system/printing/ as the lpr-linux package. Once you have downloaded and installed the software, you can add the following lines to one of your startup scripts, in order to start lpd when the system boots.

```
if [ -f /usr/sbin/lpd ]; then
   echo -n "Starting lpd..."
   /usr/sbin/lpd
fi
```

You might want to replace the path to lpd with your custom path.

However, running lpd alone isn't very useful. All it actually does is facilitate the queueing of print jobs. It does no translating or converting—that's why print filters are used. As you may know, many Linux and X applications can output and print to PostScript. This includes Netscape and the GIMP. To take advantage of these powerful applications and many others, you have to install a filter for your specific printer. Several different printer filter packages are available, so almost all popular printers are supported.

To install a filter, the first thing you must do is download the one you want. Many filters are available from the Sunsite Linux archives at ftp://metalab.unc.edu/pub/Linux/system/printing/.

The most popular print filters are **apsfilter** and **magicfilter**. In addition to the major print filters, there are many small converters and translators as well as other printer filters available for download at metalab. After you have downloaded the package you want, just follow the directions provided and install the package. (Note that metalab.unc.edu used to be sunsite.unc.edu.)

Now comes the tricky part. The lpd software reads from a configuration file called the printcap, or printer capability database. The printcap is a simple text file that holds the information necessary for lpd to output to printers. It has vast capabilities and options, but we're going to edit it in only the simplest ways right now. Fire up your favorite text editor and open the file /etc/printcap. If there is text currently in the file, make sure you don't need it (it's all commented and annotated), and comment out unneeded lines by placing a # symbol in front of the first character of those lines.

At this point, you are ready to start entering printer information into the file. Make sure you have installed your print filter correctly before proceeding.

Append this *one* line to your /etc/printcap file:

```
if=/path_to/print_filter:
```

First, replace *name* with the name of your printer. Second, replace *lpx* with the device name of your printer, which is probably lp1. Finally, replace */path_to/print_filter* with the actual path to your print filter. You must remember to change the permissions of your installed print filter so that it is executable and readable. Simply type:

```
chmod 755 /
```

Also, make sure the print spool directory, /var/spool/lpd, exists. If you want to print to a remote printer on another UNIX machine, set up the printer on that machine, then append (don't start a new line)

```
rm=remotehost:rp=remoteprinter:
```

to the previous /etc/printcap entry shown above. If you decide to do this, however, replace **:lp=/dev/lpx:** with a plain **:lp=:**. For more options to put in the printcap file, look at the man page for printcap (type **man printcap** at the prompt).

Now, restart lpd by issuing the following command:

```
killall -HUP lpd
```

There should be a brief pause, then you will be dropped back to a command prompt. If you've followed these directions correctly, do a test print: fire up Netscape and print a test page. Click on the File menu and select “Print” or “Print Frame”. Make sure the Print Command field is set to

```
lpr -
```

Of course, replace *printername* with the name you gave your printer in /etc/printcap. There is no space between the **-P** and *printername*. To print text files, you can open them up in a program that is printer-aware (such as Netscape), or you can type the following on a command line:

```
lpr -
```

Replace ***printername*** with the printer's name and replace ***filename*** with the name of the file you want to print. At this point, if all went according to plan, there should be a nicely printed piece of paper on your printer.

Sharing Your Printers

Now that you have your printer set up for yourself, you might want to consider sharing it with the rest of the network. There are a few ways of doing this. With other UNIX or Linux machines with BSD-styled print systems, just follow the directions in the previous section, and you'll be off and running. Pay attention to the specified network configuration line.

The most likely scenario, however, is one Linux machine sharing its printer or printers with one or more Windows machines. To do this, you're going to want to use Samba, the SMB implementation for UNIX, which runs quite well on Linux. After you have finished setting up the printers, install Samba on the Linux machine. Samba is available at http://www.samba.org/ and is open-source software as well.

Read through the Samba documentation to get it installed. It's a quick install, but it does require some editing of configuration files. For those of you who are impatient, here's a simple configuration file to use (locate it at /usr/local/samba/lib/smb.conf):

```
[global]
   remote announce = 192.168.1.255
   interfaces = 192.168.1.1/255.255.255.0
   netbios name = your_computer_name
   workgroup = your_workgroup_name
   printing = bsd
   security = share
[public]
   comment = Public Stuff
   path = /tmp
   public = yes
   writable = yes
```

This will export one share for use on the network, named “public”. If you have the Win95 machines set up with the same parameters, you should be able to browse the public share and look through all of its subdirectories.

Once that is done, you're ready to add the printer. The printer is added to the config file in the following manner:

```
[printername]
   path = /
   printer name =
   writable = yes
   public = yes
   printable = yes
   print command = lpr -Pprintername %s; rm %s
```

Replace *printer_spool_dir* with the printer's spool directory (I just use /tmp, but you can use /var/spool/lpd/ if you wish) and *printername* with the name of your printer (I just used hp).

At this point, restart Samba:

```
killall -HUP nmbd; killall -HUP smbd
```

Make sure you can still browse files across the network. This time, you should see a printer icon with the assigned name from the Samba configuration file.

On the Win95 clients, it would be best to install a generic PostScript printer. Then all your Win95 programs will output PostScript, and the printer filter on your Linux server will be able to both spool and print your documents as if they were local documents. To add a network printer in Win95, select the “Network” option when the Add Printer script prompts you. However, this approach doesn't always work, and you might want to use a more crude way of printing by changing the print command line in the above config file snippet to the following:

```
print command = cat %s > /dev/lp1; rm %s
```

Replace /dev/lp1 with the device to which your printer is attached. After you have replaced that line, re-install the printer on your Win95 box as the actual type; i.e., if you have a LaserJet 4L, install it as a LaserJet 4L in Win95. Note that with this method, no print spooling will take place on the Linux machine.

Wrapping Up

I hope this tutorial has helped you set up printing services in Linux. If you're fortunate enough to have a network in your home or office, you should also be able to set up the printer in question for use on the network by other computers. If you're still having trouble printing, you can check out the Linux Printing-HOWTO located at metalab.unc.edu/LDP/HOWTO/Printing-HOWTO.html. Good luck!

Michael Hughes is an honors student living in Thousand Oaks, California. His hobbies include Perl programming and snowboarding, as well as administering computers running Linux. He can be reached via e-mail at mfh@psilord.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Implementing Linux System Calls

Jorge Manjarrez-Sanchez

Issue #68, December 1999

How to create and install a system call in Linux and install an interrupt for controlling the serial port.

This article is based on my experiences in creating and installing a system call in Linux and how to install one interrupt vector to control the serial port. In one way, this is a mini-HOWTO about these two topics.

What is a System Call?

A system call (or system request) is a call to the kernel in order to execute a specific function that controls a device or executes a privileged instruction. The way system calls are handled is up to the processor. Usually, a call to the kernel is due to an interrupt or exception; in the call, there is a request to execute something special. For example, the serial port may be programmed to assert an interrupt when some character has arrived, instead of polling for it. This way, the processor can be used by other processes and service the serial port only when it is required.

The internal operation between an interrupt request and its servicing involve several CPU registers and memory segments. Briefly, a device raises an interrupt by asserting an interrupt request line on the Peripheral Interrupt Controller (PIC) which informs the CPU by setting the interrupt request pin. After each instruction, the CPU checks this pin. If it is enabled, it gets the ID from the data bus, which points to the Interrupt Descriptor Table (IDT), where a number of task, interrupt and gate descriptors are stored. The descriptor contains a selector to the Global Descriptor Table (GDT) which contains the base address to a memory segment in which the Interrupt Service Routine (ISR) resides.

Note that the CPU has suspended the process it was executing, so it has to save some information to be able to resume the process after the interrupt has

been serviced—this is a context switching. Several files are involved in this process; most can be found in the linux/arch/i386/kernel/ directory. One is entry.S, an entry point to all system calls which initializes the treatment of exceptions. Another is irq.c, which contains the functions to deal with interrupts. The linux/arch/i386/boot/setup.S file initializes the GDT, installs virtual memory, etc. There are many connections between files ending in .h and .c. You can check irq.c to see how many includes are there to get macros definitions, such as **cli()**, which clears interrupts in linux/include/asm-i386/system.h.

To follow the definition path of any function, type at your command prompt:

```
grep cli 'find / -name '*.[ch]'
-print'
```

This will search all files with extension c and h in the root directory for the word cli. Also, you can issue the command **man 2 intro** to see something about system calls.

Implementation of System Calls

There are several ways to create, install and execute a system call. The best is the one that isn't concerned with low-level details like context switching and doesn't code any routines in assembly language. This can be done through the use of the **_syscall***N* macro in the linux/include/asm/unistd.h directory; it is expanded in assembly, but the operating system takes care of details. It uses the **int 0x80** to transfer execution control to the kernel. One possible problem is this macro can expand to an existing function, so care must be taken; otherwise, you will overwrite the existing function.

In order to implement your own system calls, you should have the Linux kernel source code (first make a backup) to use as the working copy. As superuser (root), create in your home directory an entire tree copy of /usr/src/linux, as you will not have the chance to do so again. The files we will use are in *somewhere*/linux/.

Now you must choose a name for every function you are planning to implement. You can check the existing ones in your source tree at linux/arch/i386/kernel/entry.S and linux/include/asm/unistd.h. In entry.S, they are at the end, and in unistd.h, at the beginning. Checking these files will also help you get an idea of how to create a prototype of a system call. While checking, you will see that each call is associated with one number. This number is passed in the **%eax** processor register indicating the number of arguments, and each argument of the system call (a function) is passed in **%ebx**, **%ecx**, **%edx**, **%esi** or **%edi**--up to five arguments on Intel platforms. The macro definitions

corresponding to each _syscall*N*, depending on the value of *N*, can be found in unistd.h. More on the internal workings can be found in various files under linux/arch/i386/, because we will leave the “dirty work” to the operating system.

Now let's see how to implement a new system call using the syscall*N* macro in the simplest possible way. Let's make a system call **sysSum**, which accepts two integer arguments and returns the sum of the two. Also, it uses **printk**, which is similar to **printf** except that it works on the kernel level, so we will see when our function is called.

To do this, edit a randomly selected file (for example, the file linux/ipc/sem.c), and at the end, add the following lines:

```
asmlinkage int sysSum(int a, int b)
{
        printk("calling sysSum\n");
        return a+b;
}
```

Then edit unistd.h and add

```
#define __NR_sysSum     171
```

171 is the next in numerical order. In entry.S near the end, add

```
.long SYMBOL_NAME(sysSum)
```

Finally, increment by one the number that is the last line:

```
.space (NR_syscall-172)*4
```

If you don't match number and name in both files, you will get an “undefined reference to sysSum” error message. If you have a working kernel, you have to be careful only about incrementing the numbers by one and correctly writing your function name. At this point, you have added your system call; now you should get the new kernel with it. To recompile the kernel, take the following sequential steps:

```
#make config
#make dep
#make clean
#make zImage
#cat ~/linux/arch/i386/boot/zImage >/dev/fd0
```

Step 1 creates the basic kernel configuration; you can skip it next time if no hardware changes are made. Step 2 checks that any dependency between files is correct. Step 3 cleans any compilation intermediate file (object files, etc.). The last two create a compressed kernel image and copy it to floppy, so we can try our new kernel and keep the original one untouched.

Reboot using this newly created kernel to invoke the system call from a user program: simply insert the floppy disk on the drive and reboot. This simple program tests the newly created system call:

```
#include <linux/unistd.h>
_syscall2(int, sysSum, int,a,int,b)
main(){
printf("the sum of 4+3 is %d\n",sysSum(4,3));
}
```

The **include** line indicates where the _syscall definition is located. The next line says our system call has a return type of **int** and two arguments of type **int**. To compile, use the command

```
gcc -I ~/linux/include
```

to instruct the compiler to use our include file. After execution, you will see messages: first the one from sysSum, then the one from the test program.

The functions we will implement will be the basic ones needed to control the serial port using interrupts on character reception. The serial ports can't be accessed by a common user. In Linux, the functions **inb**(port) and **outb**(byte, port) exist to receive and send one byte; **inw** and **outw** do the same on two-byte data. In order to use them, you have to gain the rights by using the **iopl** or **ioperm** functions, which must be invoked as super user and will give the common user application access to the I/O ports.

The Serial Port

The serial port, called UART or RS-232, has two I/O addresses given by BIOS (on PC systems) associated with it and one IRQ (interrupt request) for each. Fortunately, they are the same as in DOS:

```
COM1 0x3F8 IRQ4
COM2 0x2F8 IRQ3
```

Each I/O port has a range of addresses to hold various support registers. COM1 is mapped in memory from 0x3F8 to 0x3FF, and COM2 from 0x2F8 to 0x2FF. See Table 1 for a description of some of them. To set one bit in any register, first read the actual value, and then **OR** with the desired value, thus preserving the other bit values.

Table 1

The Serial Port Syscall

The functions we are going to implement are the ones shown in Table 2. At this time, we will use IRQ4, but it's not difficult to use the other port or implement a select port routine.

Table 2

Listing 1

As you can see in Listing 1, we set some defines and global variables, save flags and disable interrupts to make our transaction atomic:

```
save_flags;
cli();
```

If an interrupt with higher priority takes the processor, the UART will be initialized incorrectly. We also need to indicate the routine or interrupt vector that services IRQ4. To service the interrupts, we use **request_irq** (in linux/arch/i386/kernel/irq.c) that is more or less the equivalent of **setvect** in DOS. Its prototype is

```
int request_irq(unsigned int irq,
void (*handler) (int, void *, struct pt_regs *),
unsigned long irqflags,
        const char *devname,
        void *dev_id)
```

and we call it with:

```
i = request_irq (
myirq,sioRead,SA_INTERRUPT,"sioJRMS",NULL);
if (i) return -1;
```

where *myirq* is equal to 4 (the COM1 IRQ), *sioRead* is a void pointer to the interrupt vector, that is, the routine that will service the interrupt; and *SA_INTERRUPT* is a flag that states our interrupt will be of type “fast” or non-maskable. *sioJRMS* is a name generally used to identify device drivers, but is used here to monitor the interrupts serviced by our routine by looking at the /proc/interrupts file. Once our program is running, we check this file to see if our interrupt has been set. If the value returned for *i* is 0, the interrupt vector is installed.

Next we have to set some UART initial values by using the outb function. Remember, at this time we are a superuser. After we have created our system calls, recompiled the kernel and rebooted with it, these functions will be available as an interface to the serial port in a library for every user without requiring special privileges. We use a constant, **PORT**, to identify the port address, so you can change it later.

```
outb(0,PORT + 1);     /* Disable interrupts - bit
                         0 ->0 */
outb(0x80,PORT + 3);  /* enable DLAB - bit 7 ->1*/
outb(0x0C,PORT + 0);  /* Set Divisor LSB */
outb(0x00,PORT + 1);  /* Set Divisor MSB */
outb(0x03,PORT + 3);  /* 8 Bits, No Parity, 1
                           Stop Bit */
outb(0xC7,PORT + 2);  /* Enable FIFO if UART is
                         16500+ */
outb(0x0B,PORT + 4);  /* Turn on DTR, RTS, and
                         OUT2 */
outb(0x01,PORT + 1);  /* Interrupt when data
                         received */
```

These instructions set an initial baud rate of 9600. To set to a different rate, divide 115,200 (crystal frequency) by the divisor formed by registers 3F8 (MSB) and 3F9 (LSB) when bit 7 of 3FB is 1. Now that we have initialized the UART, we can restore flags with the line:

```
restore_flags(flags);
```

We don't need **sti** (set interrupts), because it is done automatically by **restore_flags**. Next, define the routine that will service the interrupt to read a character and put it in a circular queue:

```
static void sioHandler(int myirq, void *dev_id, struct pt_regs * regs)
{
 int i;
 do { i = inb(PORT + 5);
        if (i & 1) {
                buffer[bufferin] = inb(PORT);
                bufferin++;
                if (bufferin == 1024) bufferin = 0;
                }
        }while (i & 1);
}
```

The next function is the one available as a syscall to all users:

```
asmlinkage int sioRead(void)
{
char ch;
if (bufferin != bufferout){
        ch = buffer[bufferout];
        bufferout++;
        if (bufferout == 1024) bufferout = 0;
          return ch;
        }
}
```

It will return a character from the buffer. The purpose of other syscalls is explained in Listing 1. Now we have to deal with informing the kernel that new system calls are created, using the steps mentioned previously.

In unistd.h, we put a line for each one of the newly created syscalls:

```
#define __NR_sioEnable          170
#define __NR_sioRead            171
#define __NR_sioWrite           172
#define __NR_sioEnd             173
#define __NR_sioSetDivisor      174
#define __NR_sioGetDivisor      175
```

Note that the corresponding numbers will vary depending on the total number of system calls you have. In the entry.s file, put the lines:

```
.long SYMBOL_NAME(sioEnable)
.long SYMBOL_NAME(sioRead)
.long SYMBOL_NAME(sioWrite)
.long SYMBOL_NAME(sioEnd)
.long SYMBOL_NAME(sioSetDivisor)
.long SYMBOL_NAME(sioGetDivisor)
```

and remember to increment the number in the last line.

```
.space (NR_syscalls-177)*4
```

Adding a Makefile

This time, we are not going to modify any files. Instead, we will create our library with our system calls. First, create a directory called /sio under the kernel source tree. Within it, you are going to create a file called sio.c which will contain the entire source of Listing 1 with all the includes, defines and system calls we have created. Now, in order to rebuild the new kernel with our library, we have to create a Makefile, also located in the /sio directory:

```
#Makefile for Serial Input/Output system calls
O_OBJS = sio.o
O_TARGET = siocalls.o
include $(TOPDIR)/Rules.make
```

This file will invoke **Rules.make** under our Linux source directory. Also, the top-level Makefile (the first under your source directory) will work for us. Edit this Makefile, define where the directories of sources are defined, and add our new directory with the line:

```
SIOCALLS = sio/siocalls.o
```

This appends the name of our directory to the path of source directories. Because we are using outb, we must compile with **-O** or **-O2** to enable optimization to allow the use of inline macros. Don't worry—the top-level Makefile does this. Now follow the steps mentioned earlier to recompile the kernel.

Testing the New Syscall

To work with our system calls over the serial port, make a serial cable in null modem configuration. You will need two DB-9 connectors and wire 2-3, 3-2, 4-6, 6-4, 5-5, 7-8 and 8-7 pins. Then reboot with the new kernel and use some program like the one in Listing 2 in the archive file (see Resources) as a non-superuser, and you will see you have control of the serial port using our functions. Remember to connect two Linux boxes with the cable settings described in the COM1 port.

Resources

Jorge Manjarrez Sanchez (jmanjarr@acm.org) has a master's Degree from the Center for Computing Research at IPN Mexico. He is now involved in a co-doctorate program with UPM at Spain. He has participated in several research projects mainly in the database and Internet fields and has developed a JDBC-Access type-3 driver. He spends his spare time studying Linux, Mexican History, Astronomy and leading an ACM Student Chapter at CIC-IPN.

Archive Index Issue Table of Contents

Advanced search

A Web-Based Clipping Service

Reuven M. Lerner

Issue #68, December 1999

Let LWP turn your web client into a midnight marauder.

In November, we saw how Perl's Library for Web Programming (LWP) can be used to create a simple HTTP client, retrieving one or more pages from the Web. This month, we will extend those efforts to create a program that can not only retrieve pages from the Web, but categorize them according to our preferences. In this way, we can create our own web-based clipping service, finding those articles that are of particular interest to us.

LWP consists of several modules which allow us to work with HTTP, the "hypertext transfer protocol". HTTP works on a stateless request-response basis: a client connects to a server and submits a request. The server then generates a response, and closes the connection. (If you missed last month's column, it is available here: Working with LWP. You should read that article before continuing.)

Downloading Files

We need a program that will go to a particular URL and save the contents of that URL on disk. Furthermore, we want to follow any hyperlinks in that document to collect other news stories. However, we do not want to follow links to other sites; this not only reduces the chances that we will get sidetracked, but avoids the possibility of being led astray too much.

In other words, I would like to be able to point a program at a site and retrieve all of its files on to the disk. A first stab at such a program, download-recursively.pl, is similar to the simple robot program we explored last month. It uses two hashes, **%already_retrieved** and **%to_be_retrieved**, to store URLs. Rather than storing the URLs as values in the hash, we use them as keys. This ensures each URL will appear only once, avoiding infinite loops and miscounting. URLs are placed in **%to_be_retrieved** when they are first

encountered, then moved to **%already_retrieved** after their contents are retrieved. **$origin**, a scalar variable that contains the initial URL, has a default setting if no argument is provided on the command line.

Retrievals are performed with a **while** loop. Each iteration of the **while** loop retrieves another URL from **%to_be_retrieved**, and uses it to create a new instance of HTTP::Request.

The method **$response->last_modified** returns the date and time on which a document was last modified. Subtracting **$response->last_modified** from the current time, and then comparing this result with the maximum age of documents we wish to see (**$maximum_age**) allows us to filter out relatively old documents:

```
my $document_age = time -
   $response->last_modified;
    if ($document_age > $maximum_age)
    {
        print STDOUT
        "  Age of document: $document_age\n";
        next;
    }
```

If the document is too old, we use **next** to return us to the next iteration of the **while** loop—and thus the next URL to be retrieved.

Next, we parse the contents of the HTTP response, using the **HTML::LinkExtor** module. When we create an instance of HTML::LinkExtor, we are actually creating a simple parser that can look through a page of HTML and return one or more pieces of information. The analysis is performed by a subroutine, often named callback. A reference to callback is passed along with the URL that will be parsed to create a new instance of HTML::LinkExtor.

```
my $parser = HTML::LinkExtor->new (\&callback, $url);
```

The resulting object can then parse our URL's contents by invoking:

```
$parser->parse($response->content);
```

When **$parser->parse** is invoked, **&callback** is invoked once for each HTML tag in the file. Our version of &callback grabs each URL in the file from the href attribute of each <a> tag, saving it in **%to_be_retrieved** unless it exists in **%already_retrieved**.

Finally, we save the retrieved document on the local file system. The tricky part of saving the file to disk has to do with the way in which we are retrieving the URLs—we are not traversing a tree of URLs, but are pulling URLs out in their hash order. This means the URL http://foo.com/a/b/c/ might be retrieved before http://foo.com/a/index.html. Thus, we need to ensure that the directory

/a/b/c exists on our local system before /a and /a/b are created. How can we do this?

My solution was to use Perl's built-in **split** operator, which turns a scalar into a list. By assigning this list of partial directories into an array (**@output_directory**), we can sequentially build up the directory from the root (/) down to the final name. Along the way, we check to see if the directory exists. If it does not, we create the new directory with **mkdir**. If the directory does not exist and mkdir fails, we exit with a fatal error, indicating what error occurred.

Those URLs that lack a file name are given one of “index.html”. Since this is the default file name accessed on many web servers, it stands to reason this will probably not collide with any of those names.

The end result of running this program is a mirror of the downloaded site, stored in **$output_directory**.

Sorting Through the Output

It is handy to be able to download all or part of a web site. However, our initial goal was to be able to sort through the contents of a web site for one or more phrases of interest to us.

Such a program is not very different from download-recursively.pl. Our new version, download-matching.pl, differs in that it stores only messages that contain one or more of the phrases stored in an external file, phrase-file.txt. The code for both of these programs can be found in the file ftp.linuxjournal.com/pub/lj/listings/issue68/3714.tgz.

There are several ways to perform such checking and matching. I chose to do it in a relatively simple but straightforward way, iterating through each phrase in the file and using Perl's built-in string-matching mechanism.

Since the phrases will remain constant during the entire program, we load them from phrase-file.txt before the **while** loop begins:

```
my $phrase_file = "phrase-file.txt";
    my @phrases;
    open PHRASES, $phrase_file or die
    "Cannot read $phrase_file: $! ";
    while (<PHRASES>)
    {
        chomp;
        push @phrases, $_;
    }
    close PHRASES;
```

The above code iterates through each line of the phrase file, removing the trailing newline (with **chomp**) and then storing the line in **@phrases**. Each

phrase must be on its own line in the phrase file; one possible file could look like this:

```
Linux
Reuven
mortgage
```

Once **@phrases** contains all of the phrases for which we want to search, download-matching.pl proceeds much like its less discriminating predecessor. The difference comes into play after the callback has already been invoked, scanning through the file for any new links. A site's table of contents might not contain any of the strings in **@phrases**, but the documents to which it points might.

After collecting new links, but before writing the file to disk, download-matching then iterates through the phrases in **@phrases**, comparing each one with the document. If it finds a match, it sets **$did_match** to 1 and exits from the loop:

```
foreach my $phrase (@phrases)
    {
        if ($content =~ m/>.*[^<]*\b$phrase\b/is)
        {
            # Did we match?
            $did_match = 1;
            print "        Matched $phrase\n";
            # Exit from the foreach if we found a
            # match
            last;
        }
    }
```

Notice how we surround **$phrase** with **\b**. This is Perl's way of denoting a separation between words, and ensures that our phrases do not appear in the middle of a word. For instance, if we were to search for “vest”, the **\b** metacharacters ensure that download-matching.pl will not match the word “investments”.

If **$did_match** is set to a non-zero value, at least one of the phrases was found in the document. (We use the **/i** option to Perl's **m//** matching operator to indicate that the search should be case-insensitive. If you prefer to make capital letters distinct from lowercase letters, remove the **/i**.) If **$did_match** is set to 0, we use **next** to go to the next iteration of the **while** loop, and thus to the next URL in **%to_be_retrieved**.

This all presumes a Boolean “or” match, in which only one of the phrases needs to match. If we want to insist that all of our phrases appear in a file to get a positive result (an “and” match), we must alter our strategy somewhat. Instead of setting **$did_match** to 1, we increment it each time a match is found. We then compare the value of **$did_match** with the number of elements in **@phrases**; if they are equal, we can be sure all of the phrases were contained in the document.

If possible, we want to avoid matching text contained within HTML tags. While writing this article for instance, I was surprised to discover just how many articles on Wired News (a technical news source) matched the word "mortgage". In the end, I found the program was matching a phrase within HTML tags, rather than the text itself. We can avoid this problem by stripping the HTML tags from the file—but that would mean losing the ability to navigate through links when reading the downloaded files.

Instead, download-matching.pl copies the contents of the currently examined file into a variable (**$content**) and removes the HTML tags from it:

```
my $content = $response->content;
    $content =~ s|<.+?>||gs;
```

Notice how we use the **g** and **s** options to the substitution operator (**s///**), removing all pairs of HTML tags, even if they are separated by a newline character. (The **s** option includes the newline character in the definition of **.**, which is normally not the case.)

We avoid the ramifications of a greedy regular expression, in which Perl tries to match as much as possible, by putting a ? after the +. If we were to replace <.+>, rather than <.+?>, we would remove everything between the first < and the final > in the file—which would probably include the contents, as well as the HTML tags.

One final improvement of download-matching.pl over download-recursively.pl is that it can handle multiple command-line arguments. If **@ARGV** contains one or more arguments, these are used to initially populate **%to_be_searched**. If **@ARGV** is empty, we assign a default URL to **$ARGV[0]**. In both cases, the elements of **@ARGV** are turned into keys of **%to_be_retrieved**:

```
foreach my $url (@ARGV)
    {
        print "    Adding $url to the list...\n"
        if $DEBUGGING;
        $to_be_retrieved{$url} = 1;
    }
```

Using download-matching.pl

Now that we have a program to retrieve documents that fit our criteria, how can we use it? We could run it from the command line, but the point of this program is to do your work for you, downloading documents while you sleep or watch television.

The easiest way is to use **cron**, the Linux facility that allows us to run programs at regular intervals. Each user has his or her own crontab, a table that indicates when a program should be run. Each command is preceded by five columns

that indicate the time and date on which a program should be run: the minute, hour, day of month, month and day of the week. These columns are normally filled with numbers, but an asterisk can be used to indicate a wild card.

The following entry in a crontab indicates the program /bin/foo should be run every Sunday at 4:05 A.M.:

```
5 4 * * 0 /bin/foo
```

Be sure to use a complete path name when using cron—here we indicated /bin/foo, rather than just "foo".

The crontab is edited with the **crontab** program, using its **-e** option. This will open the editor defined in the EDITOR environment variable, which is **vi** by default. (Emacs users should consider setting this to **emacsclient**, which loads the file in an already running Emacs process.)

To download all of the files matching our phrases from Wired News every day at midnight, we could use the following:

```
0 0 * * 0 /usr/bin/download-matching.pl\
www.wired.com/news/http://www.wired.com/news/
```

This will start the process of downloading files from http://www.wirec.com/news/ at midnight, placing the results in **$output_directory**. We can specify multiple URLs as well, allowing us to retrieve news from more than one of our favorite news sources. When we wake up in the morning, new documents that interest us will be waiting for us to read, sitting in **$output_directory**.

Conclusion

Many organizations hire clipping services to find news that is of interest to them. With a bit of cleverness and heavy reliance on LWP, we can create our own personalized clipping service, downloading documents that reflect our personal interests. No longer must you look through a list of headlines in order to find relevant documents—let Perl and the Web do your work for you.

Resources

Reuven M. Lerner is an Internet and Web consultant living in Haifa, Israel. His book Core Perl will soon be published by Prentice-Hall. Reuven can be reached at reuven@lerner.co.il. The ATF home page is at http://www.lerner.co.il/atf/.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Effectively Utilizing 3DNow! in Linux

Jonathan Bush

Timothy S. Newman

Issue #68, December 1999

A description of this new technology and its impact on machine performance.

In 1998, AMD (Advanced Micro Devices) released a new family of x86 CPUs that included 3DNow! capability. 3DNow! is designed to deliver enhanced performance for certain multimedia and floating-point operations. Other x86 clone CPU manufacturers, such as Cyrix and IDT (Integrated Device Technology, Inc.), also initially pledged to support 3DNow! in forthcoming CPUs. Currently, 3DNow! support is provided by IDT's most recent generation of processors (WinChip 2) as well as by AMD's K6-2, K6-3 and Athlon (K7) families of processors.

In this article, we'll describe the 3DNow! technology (especially how it impacts performance on the popular K6-2 and K6-3 CPUs) and show how to detect and take advantage of 3DNow! using Linux. 3DNow! is an exciting development; using it effectively can unleash outstanding performance by AMD and IDT processors.

What is 3DNow!?

3DNow! builds on the Intel MMX (multimedia extensions to x86) capability. Ariel Ortiz Ramirez described MMX and how to utilize it with Linux in issue 61 of *Linux Journal*, so we won't go into much detail here about MMX. Briefly stated, MMX adds eight 64-bit “multimedia” registers (MM0 through MM7), and 57 instructions that operate on those registers, to the x86 platform. Multiple short integers can be stored (packed) into each multimedia register, and the MMX instructions allow parallel computations on these packed integers. While MMX is restricted to operation on integers, 3DNow! extends the multimedia registers by enabling multiple (two) single-precision floating-point numbers to be stored (packed) into each of them. The 3DNow! instruction set includes 21 new

operations on the multimedia registers. The majority of these instructions provide fast, pipelined single-precision (packed) floating-point computation.

3DNow! capability is well-suited for fast calculation of common graphics operations such as clipping, lighting and 3-D transformations, as well as special effects involving application of physical models (e.g., fog, cloud and gravity effects). However, any application with a fair amount of floating-point computations can benefit from use of 3DNow! When used effectively, 3DNow! can increase the floating-point throughput of an application by a factor of two to four (or even more for some special-purpose applications). The increased performance results because each 3DNow! operation produces two outcomes (packed into each multimedia register), whereas standard floating-point operations by the floating-point unit (FPU) produce only one outcome per operation.

Furthermore, in the AMD K6-2 and K6-3, the MMX and 3DNow! operations have access to dual pipelined execution units, enabling up to two 3DNow! operations to execute simultaneously. Thus, up to four results can be computed per processor clock cycle on the K6-2 and K6-3. (This compares to a maximum of one floating-point result per clock cycle for the Pentium II; thus, a PII/450 has a peak performance of 450 MFLOPS (million floating-point operations per second) while a K6-2/450 has a peak performance of 1800 MFLOPS). The standard floating-point computations on the AMD K6-2 and K6-3 are not pipelined, which means there is a delay of two or more clock cycles between each concluded standard floating-point computation. Using the 3DNow! capability can turbo-charge the floating-point throughput of programs that utilize 3DNow! instructions. For computers equipped with an AMD K6-2, K6-3 or IDT WinChip2, peak floating-point performance is possible only for programs that contain 3DNow! instructions.

Getting Started

Unfortunately, few compilers can generate 3DNow! instructions for compiled code. Thus, to exercise the 3DNow! capability in programs written in high-level languages such as C/C++, FORTRAN or Pascal, it's necessary to include explicit assembly code which has 3DNow! operations. This is not difficult to do, so we will demonstrate how to use 3DNow! in C/C++ programs in Linux.

One way to determine whether a given machine supports 3DNow! is to download and run an application that identifies the processor and checks for 3DNow! capability. AMD has an application of this type that can be downloaded from their corporate web site. A practical solution for determining from within a program whether the host CPU supports 3DNow! is to use the **CPUID** instruction, which returns information on processor features and is supported by the entire x86 family. If a program determines that 3DNow! support is

present, it can exercise the appropriate sections of code which utilize 3DNow! Specifically, 3DNow! support can be determined by calling the instruction **CPUID 8000_0001h**. This instruction sets flag bits in the EDX register according to the CPU's level of multimedia support. Bit 31 of the EDX register indicates whether there is 3DNow! support; thus, CPUID sets this bit to 1 if the CPU supports 3DNow! If bit 30 is also set to 1, the CPU supports the enhanced extensions to 3DNow! available in the new AMD Athlon processor.

Some assemblers include support for 3DNow! instructions; assembly language modules that include 3DNow! instructions will be assembled without difficulty by such assemblers. However, many assemblers do not include direct support for 3DNow! In many cases, it is still possible to use 3DNow! instructions with those assemblers, although it will be necessary to define the instructions as pseudo-instructions using data blocks or emits. Fortunately, AMD's web site has a C++ header file that contains macro definitions for the 3DNow! instruction set. Inclusion of this header file can enable development of embedded assembly code within higher-level language programs. These macros specify the hexadecimal decoding for the 3DNow! instructions using the **emit** pseudo-instruction; the header file may need to be modified for certain compilers, as not all of them support emit. Under Linux, we used the freely available Netwide Assembler (NASM) to assemble code. NASM allows pseudo-instruction macros to be built using the **db** command. We have created a header file that defines the 3DNow! instructions using the **db** commands. This header file is available for download from http://merlin.cs.uah.edu/visgig/threednow/. However, NASM versions from 0.98 and beyond support 3DNow!, so the header file is needed only with older versions. Incidentally, we found that NASM 0.97 doesn't allow MM2, MM3, MM6, or MM7 to be result registers for 3DNow! operations. NASM 0.98 has no such problem.

Using 3DNow in Gradient Computation

Some applications are especially well-suited for 3DNow!, such as graphics rendering. Optimizing applications with a tiny amount of isolated floating-point operations may not be worth the effort, due to the extra time associated with coding in assembly language. There are several criteria to look at before deciding whether to use 3DNow! with the K6-2 or K6-3. First, the application should have at least a few single-precision floating-point computations grouped in one part of the program, as there is some overhead involved in switching into MMX/3DNow! mode. While in MMX/3DNow! mode, standard floating-point operations that use the regular FPU are not possible. Standard integer operations are fine while in MMX/3DNow! mode. The new K7 reportedly won't have this overhead. The MMX mode switch can also break up internal instruction pipelining, which could add overhead. The best way to minimize the impact of the overhead is to use 3DNow! in units that contain several single-

precision floating-point operations. Performance will also be improved if the floating-point data is organized in a successive, regular format (such as arrays of floating-point numbers) that enables a series of 3DNow! operations to be performed in sequence.

One application that can be efficiently implemented using 3DNow! is an image gradient calculation (edge detection), especially range image gradients. To illustrate how 3DNow! can enable efficient operations on Linux, we'll be looking at how the gradient calculation can be optimized for range and volume data. In an image collected on a 2-D grid, gradient is a measure of the local change of pixel values (e.g., pixel intensity) at a particular point; in a 3-D volume, gradient measures the difference of intensity between a voxel (volumetric element) and its neighbors. There are a variety of methods for determining the gradient. For an image collected on a grid, one way to compute the gradient is as the directional differences in value of the four immediate neighbors (to the north/south and east/west) of each pixel. When this difference is computed for the entire image, the result is that points that lie on the border between regions usually have strong gradient magnitudes. The gradient magnitudes can be viewed as an image—they look like a collection of region boundaries of the original image. The image in Figure 1 is a range image produced from laser-range data. In a range image, each pixel is a floating-point value which expresses the distance from the viewing plane of the corresponding point on the imaged object. We've displayed the image using intensities where brighter values indicate closer points. Figure 2 shows the computed four-direction gradient for the range image of Figure 1.

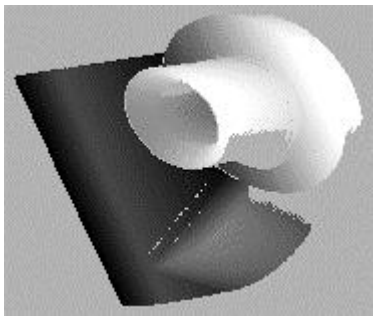


Figure 1.

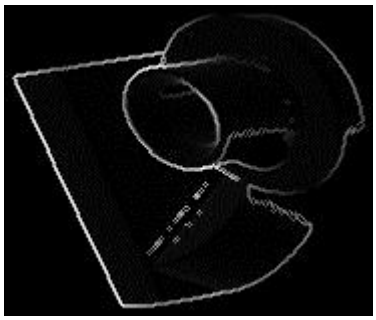


Figure 2.

In general, the gradient magnitude at a particular point is given by the equation

```
|Gradient|=sqrt((0.5*delta(
```

where delta(*x*) is equal to the change in pixel intensity in the x direction and delta(*y*) is the change in the y direction. We've illustrated the gradient for pixel P1 in Figure 3. P1's gradient magnitude is:

```
Gradient(P1)=sqrt(0.25*((depth(P2)-depth(P0))
+(depth(P4)-depth(P6))<+>2<+>)).
```

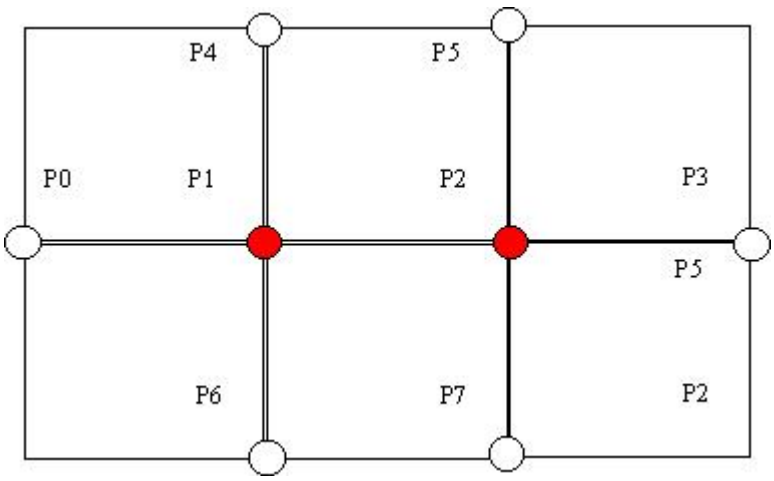


Figure 3.

3DNow! is well-suited to perform this computation for range images. Since range images can be stored as an array, points that lie next to each other on a row will appear consecutively in memory. MMX has an instruction, **movq**, that moves a “quadword” (four words—two single-precision floats) from memory into a multimedia register. This means consecutive image pixels P4 and P5 can be loaded into a multimedia register with one move. If P6 and P7 are loaded into another MMX register, we can use the 3DNow! operation **PFSUB** to subtract the contents of pairs of registers. The result of one 3DNow! subtraction will be the delta(*y*) for both P1 and P2. One more subtraction can yield delta(*x*) for these pixels. Additionally, 3DNow! operations can be used to square delta(*x*) and delta(*y*), to add them together, to apply the multiplicative factor and take the square root. The whole process can be implemented using fewer assembly instructions (and about half the execution time) than would be required for implementations using standard floating-point instructions.

We developed the assembly language function **XYGRAD** to assist in calculating range image gradients. (The code for this function can be found in the archive file ftp.linuxjournal.com/pub/lj/listings/issue68/3685.tgz.) The function processes a single row of image pixels at a time using 3DNow!. XYGRAD can be called from any C program using the prototype shown in the code. After

assembling XYGRAD with NASM, **gcc** is used to link it with a C program that utilizes XYGRAD.

Understanding Gradient Calculation Using 3DNow!

XYGRAD is passed as a pointer to the original image, called img_ptr, along with the size of the row to process. The function steps through each image row four pixels at a time. For a set of four pixels, the change in intensity in the x direction is calculated first. The quadword containing P0 and P1 is moved with movq into MMX register MM0. P2 and P3 are moved into the MM1 register. These registers are then subtracted using the 3DNow! packed subtraction (i.e., **PFSUB MM0**, **MM1**). The result is stored in the first operand, MM0, which is later squared using the packed multiplication operation (**PFMUL MM0**, **MM0**). Similar steps are followed to calculate the gradient in the y direction, with the squared difference being stored in the MM2 register. MM0 and MM2 are then added to get delta(*x*)<+>2<+> +delta(y)<+>2<+>, which is stored in the output array referenced by **result_img_ptr**. After the whole image has been processed, a second 3DNow!-optimized module is called by the C program to calculate the square root of each pixel in the resulting image. This completes the gradient calculation. The complete source code for both the C and assembly modules used in the range image gradient calculation program can be downloaded from http://merlin.cs.uah.edu/visgig/threednow/gradient/.

3DNow! Optimizations

One thing we kept in mind during the coding process is the ability of the K6-2 and K6-3 CPUs to pipeline instructions in two execution pipelines. Due to the architecture of these processors, certain restrictions apply to the pipelining of 3DNow! instructions. Namely, each 3DNow! instruction belongs to one of two subsets, and two instructions from the same subset cannot be issued in parallel. For instance, the packed floating-point subtraction (PFSUB) and packed floating-point addition (PFADD) both belong to the same subset, and therefore cannot be issued in the same clock cycle. On the other hand, the packed floating-point multiplication (PFMUL) belongs to the other subset of 3DNow! instructions; therefore, PFMUL and PFADD instructions can execute simultaneously, provided there is no operand dependency between them. It's beneficial to interweave instructions from each subset as much as possible to increase the likelihood of parallel computation. Most integer MMX operations do not have such a restriction on the K6-2 and K6-3; achieving optimal floating-point performance does require a little more consideration by the programmer.

Gradient Calculation Performance

Our 3DNow!-optimized gradient-calculation programs delivered excellent performance. We conducted tests of the programs on a dual-bootable PC with a 300MHz AMD K6-2 CPU to determine 3DNow! performance in both Windows and Linux environments. On Linux, we used the Netwide Assembler (NASM) version 0.98 to assemble the assembly modules. We used GNU C (gcc) version 2.7 to compile a C driver and link the driver to the assembly code. On Windows, we used a popular commercial C compiler to assemble, compile and link the assembly and C codes. The Windows assembler/compiler did not recognize 3DNow! instructions, so we had to include the AMD header file that defines 3DNow! using the pseudo-instruction macros. We tested the 3DNow!-optimized gradient calculations on several range images and 3-D volumes.

Tables 1 and 2 show performance results for two representative data sets. In Table 1, the average CPU times for execution of the 3DNow!-optimized range image gradient calculation in Linux and Windows environments are shown. (Times are averaged over 2000 trials for a 240x240 range image.) By comparison, the 3DNow!-optimized code ran about nine times faster under Linux than unoptimized standard C code which computes the data according to the same pattern of pixel access, i.e., the unoptimized code was implemented purely in C without use of 3DNow! and was compiled without any compiler optimizations. When the standard C version of the gradient was compiled with maximal optimizations by gcc in Linux (using optimization switches of **-O2 -ffast-math**), it was still more than three times slower than the 3DNow!-optimized code. This time, improvement is significant; the 3DNow!-optimized version of the range image gradient calculation can be done in real time at frame rates under Linux. The performance improvement was nearly as impressive for the volume gradient computations—the times shown in Table 2 are for a 72x256x256 volume. The 3DNow!-optimized volume gradient executed more than 2.5 times faster than fully optimized standard C implementation, and better than 4.5 times faster than an unoptimized standard C implementation.

Table 1

We've also tested the performance of the standard C range image gradient implementation (i.e., the implementation without 3DNow! code) on a Pentium II/333 running Linux. Using full compiler optimizations, the 240x240 range image's gradient was computed in 17% more time on the PII than on the K6-2. Thus, we estimate that the K6-2 can calculate range image gradient under Linux about 30% faster than a comparably clocked PII. Of course, we should add that it's generally easier to develop modules in C than in assembly language.

Table 2

Conclusion

3DNow! is an exciting development for desktop computing, offering the potential for significantly improved performance for many applications. 3DNow! can be effectively utilized in Linux using NASM and gcc. For applications that involve floating-point calculations, especially those where speed is critical, incorporating modules that utilize 3DNow! instructions can unlock outstanding floating-point throughput in popular CPUs such as the IDT Winchip 2 and the AMD K6-2, K6-3 and Athlon processors.

Jonathan Bush (jbush@cs.uah.edu) recently received his B.S. degree in Computer Science from the University of Alabama in Huntsville. He completed much of the work described in this article as a National Science Foundation Undergraduate Research Experience Scholar. He is currently a graduate research assistant in the Department of Computer Science.

Timothy S. Newman (tnewman@mailhost.cs.uah.edu) is an Assistant Professor of Computer Science at the University of Alabama in Huntsville. When he's not teaching, he can often be found conducting visualization and imaging research on Linux boxes.

Archive Index Issue Table of Contents

Advanced search

Focus on Software

David A. Bandel

Issue #68, December 1999

mason, si, system-info and more.

The time is on us once again. The “feature freeze” for 2.3 was just announced. By the time you read this, it will be down to last-minute testing and making sure all is well before final release. What's new other than more device drivers? I don't yet know everything new, but I do know that once again, I'll need to learn new firewalling software. For 2.0, it was **ipfwadm**--not bad, but no fine-grained control. For 2.2, it is **ipchains**--I liked the control, but heard many complaints about its complexity, and I found very few configuration tools for this beast. So I'm off to download and compile the latest kernel to test the new **netfilter** that will be 2.4's packet mangling software. Here's hoping for configurability and simplicity in one package.

mason: http://www.pobox.com/~wstearns/mason/

This aptly named software is, you guessed it, a firewall configuration program. Basically, **mason** learns about the traffic passing through your gateway (soon to be your firewall) and records the traffic so you can build a firewall brick by brick (or chain by chain, as it were). The recording is done in the form of a line that can be used by mason or by the **ipchains-restore** script. When the software fires up, it checks what type of system you have: if it is a 2.0.*x* system, it will use ipfwadm; if 2.2.*x*, it will use ipchains. The new netfilter software rules should not be significantly different from ipchains, and support will be added before the 2.4.*x* release if it hasn't been already (some of the code was in place but disabled in the version I tested). The software does require you to review the rules, so you do need to be able to read and understand them to decide which rules to keep. It requires bash, ipchains or ipfwadm, and a kernel built with firewall support.

si: http://si.netpedia.net/

This command-line system information utility will fill pages. **si** will tell most folk more than they ever cared to know about their system, what resources (IRQs, DMAs, etc.) are being used, what programs are running, how much memory they're using, etc. The information can be obtained by other programs, but it will take a few. In fact, I'm not sure what more information you could get or want. While I haven't verified it, I suspect this program is reading a good part of the /proc tree to return all this information; at least, it matches the information I know to be available, just not as easily readable in /proc. It requires glibc.

system-info: blaine.res.wpi.net/files/system-info.0.7.tar.gz

Going from information overload to almost underwhelming by comparison, this utility will provide one page of information nicely formatted in HTML—great for putting something up on a web page. I looked, and while it had a fair amount of information for only one page's worth, it was innocuous enough. I would feel safe putting this on a public web page, whereas the utility above is more information than even a wannabe cracker would want (or need). It requires Perl.

DNS sleuth: atrey.karlin.mff.cuni.cz/~mj/linux.html

This little jewel is a DNS checker. With both a command-line interface and a web interface, **sleuth** will check whether the configuration of your DNS complies with the RFCs. It will give you warnings for some things and errors when it sees something completely wrong. The best part is it will tell you what is wrong and reference the RFC so you can see for yourself why it's bad and how to fix it. No more guessing if it's correct or not—fast and thorough. It requires Perl and the Net::DNS Perl module.

landb: http://avenir.dhs.org/landb/

Have a large LAN? Thankfully, I don't any more. But if I did, particularly one that spans buildings (much less floors), and typically two or more /24 (class C) networks, I'd be using something like this database to keep it sorted. It really is overkill for a small network, though. I think I'd add a few comment fields to hold a name and number or two for problems. Makes a nice complement to a resource manager like MOT (Ministry of Truth) or IRM (IT Resource Manager). It requires Perl, CGI, DBI, DBD modules, MySQL and a web server.

yafc: http://www.stacken.kth.se/~mhe/yafc/

yafc is yet another FTP client. You may be thinking, “I already have both graphical and command-line FTP tools, and **ncftp** (a command-line client to which this is a competitor) fills the latter niche nicely.” However, the nice thing

about competition is the newcomer has to have something that works better than the incumbent, or otherwise why bother? Well, this one has—at least for me. Side by side, I found yafc easier to use (important even to a command-line-junkie like myself) and better designed. It has a few parameters you can set, like **cache** and others. It requires libncurses, libreadline and glibc.

august: http://www.lls.se/~johanb/august/

It's been a while since I looked at any kind of HTML markup editor, and I don't remember them being all that friendly or easy to use, so my HTML editor of choice has always been **vi**. Now, you've probably guessed I'm not much of a webmaster (it's true, I'm not)--I'm into substance over form. About the only thing I didn't see in **august**, but would like to, is some markup selections for PHP. It requires Tcl/Tk.

David A. Bandel (david@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. Co-author of Que Special Edition: Using Caldera OpenLinux, he plans to spend more time writing about Linux while relaxing and enjoying life in the “Crossroads of the World”.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #68, December 1999

Readers sound off.

Practical Linux

I am at the crossroads, so to speak. Don't get me wrong—I'm a fanatic Linux user, but... First of all, some background: I work for Futurekids (South Africa), and our mission is basically to teach children and adults computer mastery skills. Currently this is done entirely on MS platforms. I started to search for alternative budget systems similar to what we use on MS, but had no luck. It seems the only use for Linux in the classroom is for Internet access and file/printer sharing.

This brings me back to the underlying problem. If we truly want to promote Linux, we must start with the younger generation—the adults of tomorrow. If we can work together to implement a solid education program based on the Linux platform, I believe we will have success. The net result? Children use Linux at school—they get frustrated with Windows at home—and eventually more and more home Windows systems migrate to the Linux platform. But we need this educational system first. To start, the software developers must begin producing "kids" programs—and please, not another flash card program—but programs that make use of multimedia to teach children how a mouse works, how the keyboard works, then progress to Encarta-like encyclopedias, etc.

Dear Linux Developer Community—there is your challenge!

—Nico Coetzee nicc@mweb.co.za

Product Support

I have been using computers for many years, so I am no stranger to the difficulties associated with installing a new OS. But to make claims in your magazine that so far are false, such as the ease of loading Caldera, has put me

off both your magazine and Linux in general. I have tried to contact the company: the person at the 1-888 number could easily be replaced by a tape machine, technical-support lines requested I pay more money to make their product work, and I have received no response from e-mail support. I write to you because I read a review in your magazine, and dopey me, I thought someone there used the software before they wrote about it.

—Michael Brooks AZTowGuy@aol.com

Sorry you had problems. In actuality, at least four people here at the office, including the reviewer, used the Caldera LIZARD install successfully with only the one problem discussed in the review. Since that time, we did try to install it on an old machine with a strange configuration, and LIZARD hung up. We switched to the LISA install and it worked fine, installing everything off the CD-ROM without a problem. Took longer than LIZARD, but was still fairly easy. Writing an easy install that includes all possible configurations of hardware, both old and new, is not an easy task —Editor

The Revolution Continues

First off, I would like to say good job with your magazine. I always look forward to the new issue hitting our shelves in my store. I work for Barnes and Noble booksellers, and that's what the other part of this letter is about. I wanted to inform you that due to the support of our Customer Relations manager, and of the store in general, we have started a Linux Users Group through my local store. We meet in the store once a month (the third Tuesday of every month, for those of you in the Lakewood/Long Beach, CA, area). It has quickly grown to be the most popular group in our store, even surpassing the regular reading groups. I hope this trend continues and we see more user groups popping up. Thanks again.

—Jason Lundy lcghent@yahoo.com

Is KDE the Answer?

I have to disagree with Phil Hughes' position in “Is KDE the Answer?” in October's *Linux Journal*. At the end of the article, Phil states that we should all jump on the KDE bandwagon because KDE is further ahead and is the default on more distributions. He also states that one standard GUI is better than two for appliance users.

I think this is wrong. First of all, there is something to be valued in diversity, as diversity breeds strength. If one day KDE runs into a brick wall and cannot further advance, we'd still have GNOME. Furthermore, if we all jumped on the product that was furthest ahead, Linux would not exist today.

Second, I see no advantage to giving appliance users (or newbies, or any other user) only one interface. While this may require less mental exertion on the part of the user, it will also lead to a bland, homogenized world. A different, novel interface may actually appeal to some of the appliance users.

Phil also mentions offering “one standard GUI rather than two”. This is like saying the US has only two political parties. I use neither KDE nor GNOME, because neither works well for me. Just try running either on a slower Alpha with a TGA card, and you'll know what I mean. Instead I use **icewm**, which I think is far superior to either. I've used other window managers such as **mwm**, **amiwm** and **twm**, and find all of them far faster, more stable and in almost all ways superior to KDE and GNOME, especially on low-end hardware.

By saying that we should all jump on the KDE bandwagon, Phil is saying that we should all fit ourselves into KDE's mold rather than finding and developing the tools that fit us. In my book, KDE is not the answer.

—Richard Griswold griswold@acm.org

Stability

I always look forward to reading my issue of *LJ* each month. Great magazine; thanks.

The reason I am writing is to share a concern regarding Linux API stability. I enjoy system work too, but the fact is my company sells applications, not operating systems. If I am to produce shrink-wrapped applications, user and kernel APIs must remain stable. I just worked my way through our several drivers to get them from 2.0.* to 2.2.* series kernels, and now I see the driver API is changing again in 2.3.*. It is a terrible temptation to keep improving, but if it works, let's leave it for a while. And maintaining old APIs is not a long-term answer either—that's what makes UNIX systems so fat, all the baggage they carry around.

Thanks for listening.

—Elwood Downey ECDowney@ClearSkyInstitute.com

Archive Index Issue Table of Contents

Advanced search

Re: gcc performance on NT

I just read the *LJ* article on the performance comparison between Linux and Windows 95/98/NT. In the article, gcc and pgcc on Linux were compared to VC and CodeWarrior on Windows. The results showed that gcc and pgcc were faster, and the article made the claim that this means Linux is faster.

I'm curious—would Linux really be faster if gcc were run on Windows? How does the program perform when compiled with DJGPP, Mingw32, or Cygwin? Also, how would lccwin32 (the lightweight c compiler) perform? —Paul Brannan

Thanks for your note.

You are definitely correct that performance is a function of many things—the compiler, the memory, the motherboard, the CPU, the operating system, etc. I think we were able to isolate performance to primarily the OS and the compiler in our study (although I think we also were able to do some comparison of CPUs in the study). We've tried to emphasize in the article that it's the Linux/gcc or Linux/pgcc combination that is faster than the Win/VC or Win/CW combination.

Here's one datapoint which we didn't present in the article. We have benchmarked the application on some Sun Solaris machines which had gcc and Sun's C compiler. On the Suns, the Sun C compiler produced binaries that were about 30-40% faster than gcc.

Also, gcc—on Sun/Solaris and on Intel(or AMD)/Linux—takes very little advantage of advanced architectural features. For example, gcc typically does not generate MMX instructions, and it never generates 3DNow or SSE instructions. On Sun, gcc does little to take advantage of Sun's Visual Instruction Set. Under Windows, Code Warrior CAN issue MMX and 3DNow instructions, and Visual C can issue MMX instructions. Moreover, both Code Warrior and Visual C have customization settings to take advantage of Pentium and Pentium II (and now, probably Pentium III) architectural features. Code Warrior can also generate code that takes advantage of K6-2 architectural features (features besides only 3DNow!). Therefore, we believe that it is unlikely that gcc is generating code which is closely targeted to a specific architecture—at least, the code would not be expected to be targeted as well as Code Warrior and Visual C since those two can optimize for the specific architecture. Therefore, we believe that Linux is indeed likely to offer a more efficient execution environment than Windows.

I'd also like to note that we have considered comparing assembly code generated by gcc and the commercial products, but it's been suggested to us that the terms of licensure for most of the commercial compilers imply that “reverse-engineering” activities are not acceptable uses of the licenses. We felt that comparing assembly code could be viewed as a reverse engineering activity, and so we have made no attempt to compare the assembly code being generated by the compilers. Instead, we've done what we could do, which is just focus on the bottom line and observe which complete environment produced the best execution performance.

However, again let me state that you are correct that it's not completely possible to decouple the compiler and the OS. That was a point that we tried to make in the article, although possibly we could have made even more explicit and strong statements on that issue in the article. That's one reason that we tried to point out that at least the gcc/LInux (or pgcc/ Linux) _combination_ was more efficient than VC/Win or CW/Win. We find that to be fairly remarkable - a completely freeware solution can actually operate more efficiently than a commercial solution (especially given the enormous amount of effort on the commercial products by some very talented individuals) and we thought that the computing community needed to be presented with our study's results.

One last word. We did not test this particular application using gcc under Windows. However, in the past we have compiled other (smaller) applications using gcc under Windows. We have compared those applications' performance with the performance of code compiled with commercial compilers and observed that the commercial compilers produced faster code. Our experiments were not rigorous, however, and were on a limited set of examples. Due to those experiments, we thought that it was unlikely for our application to run as efficiently if it were compiled using gcc under Windows than if we used something like Code Warrior under Windows, which we know produces good-performing code. Also, I believe that our version of DOS gcc is a 16 bit version, and we thought it would be unlikely to be able to produce reasonable execution times for our application.

But, I like your suggestion of trying additional compilers, which could make for a strong case that Linux's computational performance is superior to Windows. I will try to benchmark the application using gcc and a few other Windows world compilers. I'll get back with you with results. If you don't hear from me within a week, could you drop me an email. I'm pretty busy and sometimes I put things off for a while and then don't get back to them for some time. A reminder would help me get around to it quicker.
—Dr. Tim Newman, tnewman@cs.uah.edu

Comments on November 1999 issue
Two observations:

1. PPP:
I suspect that the point of Gerry George's question was missed in the answer to his PPP question. For purely local access, one can assign a private IP address to the interface by default. A private IP address is one selected from the ranges listed in RFC1918 and guaranteed to be unroutable on the public Internet, such as 192.168.x.x.

If security is not a huge concern and users can generally be trusted once they provide their name and password to satisfy the ordinary login process, then one can insert “ipcp-accept-remote” into the “options” file which will be read by “pppd”, usually in “/etc/ppp”. This tells “pppd” to accept whatever the remote peer wants to use as its IP address.

If the answerer wishes to force the caller to supply IP addresses for the answerer's interface, then it should set its own address to 0.0.0.0. Note that some PPP software, such as Windows Dial-Up Networking, may be unable to cope with such an answerer.

One should also include the “lock” keyword in the “options” file so that a dialout does not occur while someone is dialed in. Also, both the dialout script and “getty” should reference the same interfaces using the same names, such as “/dev/ttyS0”. The use of the “/dev/cua0” form of interface name should be avoided for this purpose.

2. Network Time Synchronization:
In response to John Morley's question, it is important to emphasize that there are different types of network synchronization. First, in some cases, it is more important that colocated machines be precisely locked together than that they be accurate relative to some external reference. Many protocols, such as NFS and Kerberos, benefit from tight local time synchronization among cooperating machines.

Since real NTP is relatively straightforward to configure and totally free, there are really no good reasons to rely on “rdate” or “timed” methods. The issue is not so much absolute accuracy as fault tolerance. All programs which I know about that use the RFC868 TIME (port 37) protocol simply query one server and believe whatever they are told. Personally, I have seen widely used public time servers at major universities give out junk answers on occasion; I once had a machine set its clock to February 2037 when such as server began sending all zeroes.

With NTP, attempts are made to measure network delays and apply some degree of sanity checking to the results returned. In addition, if a web of NTP servers are configured together, then they will tend to stay closely locked together around a consensus mean time. NTP also allows using MD5 and other cryptographic authentication so that only the machines configured together will be trusted to nudge each other's time.

There are excellent freeware Windows clients which implement SNTP, a simplified client-only version designed to be used to query a real NTP

server. I recommend Dimension4, a freeware SNTP client for Windows 9x/NT, available from Thinking Man Software:

http://www.thinkman.com/~thinkman/dimension4/

For extremely casual Unix use, such as might be appropriate for a desktop workstation, the “ntpdate” SNTP client program included with the NTP distribution is a much better option than “rdate” or “netdate” methods.
—Mike Bilow, mikebw@colossus.bilow.com

Reply to your post
Anonymous replied to your post 'my editorial in *LJ*' at the site: *Linux Journal*. The reply was:

Well, for one thing your editorial made me glad I have decided not to renew my LinuxJournal subscription. There are several very important reasons why GNOME should be the standard linux desktop, and most of the have nothing to do with GNOME itself. a) The licensing issues. I know some people have tried to call this one of after the QPL, but it is definetly not resolved. Libraries should be LGPL anything more restrictive is a mistake. The toolkit put under the LGPL is important because it, for all practical resons, places the ownership of the toolkit in the hands of the community which means that all who helps develop from can potentially benefit equally. Another aspect of this is that Linux success is and will always be about the OS beeing a free platform for all, even for people who in my opinion makes the wrong choice and release the software under proprietary licenses (which most of these “open-source” licenses qualify as.) John “Maddog” Hall, the leader of LI, made a statement in a interview some time back when asked if he thought all software should be free, where he answered that the most important thing is that the OS itself is free. If Qt was to be accepted as the primary toolkit this would no longer be the possible. (Note, I don't know wether Maddog shares this interpretation.)

This is the single most important reason to reject Qt, and therefore unfortuantly also KDE.

b) The language choice. The fact that Qt and KDE is so C++ based, makes it very difficult to make wrappers for other languages. Which would lessen the attraction to Linux for many developers. AFAIK the only wrapper currently available is Perl althought there might be a few others. Where Gtk and Gnome on the other hand uses stanard C and therefore already have gotten wrappers for C++, Objective C, Python, Pearl, Ada and many more. Even more wrappers are under development to support languages such as Java and Pascal. The use of C as the base language should be of little consequense for C++ developers with great C++ and Gtk based projects like Mozilla and AbiWord to show the way.

c) The last reason I will give at this point is simply that GNOME is to great to be discontinued, even though the project was started around a year later than KDE is has already caught up, and in many areas surpassed KDE. The last few areas, such as some rough edges caused by product immaturity, is being taken care of with the 1.0.50 release being readied these days.

Okay an extra bonus reason: GNOME is the official GNU desktop and without GNU there would be no Linux. Sincerely,
—Christian Schaller, frostking@linuxrising.com

---

hurray non-hatemail!

To: Jason Kroll
Keep up the great work in *Linux Journal*. It was because of your cool articles that I bought and subscribed to the mag!!! Sincerely,
—Richard Tricoche, richard.tricoche@rpawireless.com

Floppy Formatting

John C. Burgess wrote, in Letters to the Editor, in regards to being able to format floppies as a non-root user:

```
chmod 777 /dev
chmod o+w /dev/fd*
chmod 777 /usr/bin/fdformat
reboot
```

Anyone following this advice can compromise the security of their system. An explanation follows.
First, there is usually a group called floppy in the group file. the /dev/fd* files are also usually members of this group. A user can be added to this group (usually in /etc/group,) and they will have access to the files that groups owns the next time they log in. This means they will have write access to the floppy block devices the next time they log in.

So instead of giving global access to the floppy devices, add specific users to the group which has write access to the floppy devices. Giving global access to the floppy device allows anyone logged in to change any of the files or blocks on the floppy at whim.

Don't give global write access to the fdformat program (anyone who logs in can change the nature of the program and insert bootloader viruses on the floppies! - a rather hysterical example, that is.) I.e. don't ever type chmod 777 on any system binaries, libraries, configurations or any other files.

As for rebooting, your login shell will have to be restarted, but nothing else should need to be restarted, esp. the kernel. I typically add a user to the group file, and then I have to restart the login shell before the new permissions are recognized.

Another piece of errata: I've heard most recently people referring to root access as `from any directory except root.' This is a misconception. The root directory, /, is the first mount point in the Linux system, and is the start of the filesystem path (hence the chroot command.) Root access, as the user, is merely just another login, only the root user has access to every aspect of the system, including the running kernel! Don't use root!

For John C. Burgess: whatever `gurus' you asked this about were flat wrong, but the solution you came up with, while testimonial of the flexibility and power of Linux, was probably the worst solution you could ever force on yourself in the long term.
—Christopher Rhodes, christohper@evergreen.net

Correction (?)

In your reply to “Memory Error, Parallel Computing” in the October issue, one possibility for the first question was overlooked. Some systems (like Compaq's 9546) limit the amount of memory reported to the kernel to slightly less than 16M. A simple fix is to use the boot option:

```
mem='XXM'
```

where XX is the amount of memory installed in the system.
—Kevin, kevin@obone.net

Word Perfect 8 for Linux is unusable

I just downloaded and installed Word Perfect 8 for Linux on a RedHat 6.0 system. This product is not ready for prime-time.

The download and installation instructions were, on some points, vague, and even misdirecting. This is not good.

More importantly, all application graphics show as garbled checkerboards, making the Word Perfect 8 application unusable. Even more importantly, according to tech support, this is the behavior of the commercial product, and they say they have no idea what the cause is or how to fix the problem.

I was hoping to find a familiar word processing application for my Linux oriented clients, but I certainly can't recommend WP8.

I'm sorry your company is presenting applications for the Linux operating system in this light, since I experience a superior level of reliability with most Linux applications. You can see what this says about Corel, of course.
—James Feeney, james@topaz.nurealm.net

I would greatly appreciate your advice. (fwd)

To: Lydia Kinata
You have my dream job, I dream of making layouts for a magazine. I plan

on Majoring in Graphic Design and Advertising, with a minor in Business. My only problem is that I don't know where to go to college, and I don't know if those are the right majors and minors. I would greatly appreciate any of your advice.
—Magoo25@aol.com
Well, the San Francisco School of Design has a great reputation in the business, as does the Rhode Island School of Design. Other than that, most universities will have some kind of design program, and there are smaller specialty schools. We have a junior designer here in the office who just graduated from the Art Institute of Seattle.

Your planned majors sound good, you'll have a broad base there. The business classes will help if you decide to work for yourself.

My education includes 2 years of Design from the now-defunct Factory of Visual Arts in Seattle. I can't say I learned anything relevant to doing magazine layout there, other than general design concepts. My computer design and layout skills are entirely self-taught, and I'm learning all the time. I *can* say that you should plan to become an expert at Adobe PhotoShop, Illustrator and InDesign, as well as at Quark Xpress. Beyond that, some CAD or 3-D Rendering software would be really helpful. Hopefully some or all of these applications will be available for Linux soon.

And, most importantly: a lot of design schools teach nothing or next to nothing about real-world design for print. Learn how to scan art, set up graphics for print, determine the correct file format to use, trap, and do color separations. You'll make mistakes as you gain professional experience, but try to avoid the “oops, that won't print” or the “oops, I don't know how to plan for output resolution” mistakes that happen to new designers. Also, learn to make your printer your friend (not your Epson or HP, rather the press man at the print house). You can learn more from a good pressman than from almost anywhere else, and a lot of designers don't realize how much time and money can be saved by talking to the pressman before beginning a design.

Well, that's it. Good luck!
—Lydia Kinata, info@linuxjournal.com

Correction to “Chosing the Right Commands” in the Oct Linux Journal

I read the question posted by Mark Foucht in regards to mounting CDs in your October issue. The answers given in regards to how to share the mounted images were correct, but one detail that was left out was HOW to get those images so that they could be shared. I wasn't sure if Mr. Foucht's question was more about how to actually make/mount the images, or how to share them.

To make/mount the cd images:

- The kernel on the box that will be sharing the images will need to have support for the loopback device either compiled as a module, or compiled into the kernel
- This machine will need to have the the /dev/loop* devices created if they do not already exist. This can be done by going to the /dev directory and issuing a “MAKEDEV loop” command.
- The loop.o module will need to be loaded with lsmod if it was compiled as a module. On my Mandrake 6.1 system this module is located in the /lib/modules/2.2.13-7mdk/block/ directory. If you did not compile the loopback support as a module, you will need to install the kernel, re-run lilo and then reboot to activate the new kernel.
- You will then need to create the iso image of your CD by mounting it and using the mkisofs command. For example:

  ```
  mkisofs -r -o cdrom0.iso /mnt/cdrom
  ```

  will create the file cdrom0.iso with Rock-Ridge extentions in the current directory from the filesystem mounted at /mnt/cdrom.
- Once this is done, you need to associate the a loopback device with the ISO cdrom image that you've created and then mount it. To do this for the iso image “cdrom0.iso” at the mount point /mnt/cdrimage0 this is done by:

  ```
  losetup /dev/loop0 cdrom0.iso
  mount -t iso9660 /dev/loop0 /mnt/cdimage0
  ```
- To umount the image you need to issue the commands

  ```
  umount /mnt/cdimage0
  losetup -d /dev/loop0
  ```

I hope this, along with the answers supplied will give Mr. Foucht a more complete answer to his question.
—Steve Fuller, sfuller@fuller.des-moines.ia.us

Pictured

hello,
there are two corrections to the picture on page 14 of your october magazine.

1. as could be easily seen the text should be Pictured (from left to right) are Robbie “maddog” Honerkamp, Steve “maddog” Lewis, Jon “maddog” Hall, Greg “maddog” Hankins, Antoni “maddog” Dabek, and Reg “maddog” Charney.
2. as none of the guys has a proper shirt and no suspenders are visible so none of them is a real unix guy.

—engelbert gruber, engelbert.gruber@ssg.co.at

Is KDE the Answer?

Hello,

I just read your article on *LJ* issue 66, and I would like to give my opinion on it. I realize you will get plenty of “opinions” about it, so I'll try to be short, even if it's a complex matter.

The first thing I thought, trying a KDE beta some time ago, was “oh! Finally a decent window manager!” but I'm sure this can be only a so personal point of view, maybe because I've been using Windows more than X-Windows; after all this can be the thought that so many ones have had. One of the major killers to Linux diffusion, in my humble opinion, is after all it's appeal: you can do wonderful things with it, but users will have more troubles configuring /etc/printcap than a GUI equivalent. The point you raise, about not skilled users to think they gained “computer literacy” is after all sadly true. I found so many small ISP, and sometimes even small informatics departments in schools, having a SysAdmin who didn't even protect TCP ports like fingerd to outer world, giving away full users' names. From my point of view these guys are Windows NT administrators that just discovered Linux, and when they succeeded with their first internet connection started thinking they could “master the internet”. But this example addresses something that's not mentioned in your article: the level of literacy required by a computer user.

Your words appear more a crusade toward the pureness of computing, than pointing to the users' lack of skill. I've heard and read similar things before, and it's so sad to see that what's presented as an “open movement” after all wants just to be a close community of experts: unless you are dreaming of a world where everyone knows the most about operating systems.

And that close minding on open software happens, ironically, even in teams like the KDE one, let me tell a short story. When the KDE was out as a Debian package I installed it, and found that the FHS adopted was giving major troubles trying to configure and compile older software, even if there was a 'configure' script made with autoconf. I contacted Stephan Kulow, the maintainer of the Debian packages, and he (kindly?) answered that he wasn't not susrprised an Italian like me didn't understand about autoconf, and he declared the argumentation over.

So I consider such comments as a lack of reality, because the daily users of a computer, like people doing just word processing, or charts and invoices, can't be expert users of an operating system. And they will never be, considering how fast software products are evolving. Then of my questions is: are we cutting out 90% of computer users just to feel the ones who hold the real power of computing? If this is the purpose of the open source movement then I don't even want to be in it, no thanks. I don't want to be using an operating system that discriminates what users are good or bad ones: wasn't that a prerogative of the already estabilished commercial OSes?

I keep seeing software produced by wonderful programmers that has a hard to use interface, hard to configure, and in the end justified as “only for real computer people”. This is crazy, keeping things hard to use only for a small number of adepts. There's nothing you can call “open” in all this.

I'm developing embedded systems for more than ten years, designing them from hardware up to high-level system interface. With all this I mostly worked with C language, and a lot of assembly programming on so many microprocessors, and that lead me to keep liking more C to C++, because on small systems software has to be predictable with memory usage and real time response. Despite all this, another thing that I thought looking at how KDE is programmed made me think “Hey, this is how Windows should have been made”.

If you want to make a new world (since we already have one running), you have to get your hands dirty with something, and you can't ignore it.

I really don't understand what is making “internet safe again”: using KDE leads to making internet unsafe? Maybe it's because I'm european, so I'm not as close minded and bigot as many americans I see, after all we had a different history, but... I'm really not the kind to go out with a sign saying that the end of the world is soon to come, or asking for a holy war.

Ok, armed with console applications you can do more than with a GUI, maybe even I can do a little magic with it, but why the heck has _any_ Linux user to do so? Should it be one of those proof of adultness to achieve like in a primitive culture?

You address a real problem with things like paper size treated to be A4 instead of 'letter', that's something for developers to fix. I know very well that kind of problem: I've been fixing for many years software that was using 'letter' instead of A4, and inches instead of centimeters. So this just shows that can exist blind programming on both sides of the pond.

So you don't see any progress in '99 programming compared to '76 one? I'm not surprised. Can't see any way you could.

One final note. *Linux Journal* has now a better appeal, with more colored pages, better organized pages and sections: doesn't it make readers think they read it better, while this is totally wrong? I can't see how a publisher did let this happen, as we know that in the end leads guys to think they know how to manage a magazine (after they badly drove their car all the way to come telling you, of course). Luckily the magazine has rare spots of this catastrophism in it, otherwise I wouldn't have a good reason to keep being a subscriber. Open your mind before you open your software source, or you will give for free something that no one needs.
—Lou Cyphre, LouCypher@cheerful.com

Floppy formatting

I saw your comment about floppy formatting in the October issue of *LJ*.

In fact, the following is what I did (uh, back in 1995):

- create a “floppy” group; put the users you trust in that group
- chgrp floppy /dev/fd*
- chmod 660 /dev/fd*

Note the following:

- a “floppy” group is used (MAKEDEV does this by default!)
- /dev/fd* devices should have 660 permissions, not o+w (world-writable) (unless you don't believe in “Murphy's Law”)
- a reboot is *not* necessary

Even if you don't use a “floppy” group and just o+w the devices, a reboot is still not necessary :-)
Regards,
—Ambrose Li, acli@acli.interlog.com

Is KDE the answer?

In your October 1999 editorial, you mention that KDE being configured for A4 paper size by default as a problem, even going as far as considering it a bug (though you _do_ point out it can be recompiled otherwise).

May I recommend that you do what's right and upgrade towards this world-wide standard format, instead of perpetuating that old, deprecated remnant of American industrialism know as US Letter?

The DIN paper formats may have started in Germany, but they have now become a world-wide standard, not only in Europe but also in Africa, Asia (except Japan) and South-America.

The standard itself is rather clear and logical and it includes just about every kind of stationary out there: paper, envelopes, business cards, invitation cards, passports, ID cards, etc. etc.

Like the metric system, the DIN stationary format is one of those standards America really ought to upgrade to. The question is not why but when, as both standards have become unavoidable.

Could Linux users lead the way, by implementing the DIN formats and adopting them from now on, even in America?
—Martin-Eric Racine, q-funk@intellitel.com

About “Is KDE the Answer?”

Dear Mr Hughes,

I am probably about the 999th person to protest to you about your editorial “Is KDE the Answer?” in the most recent *LJ*. But I hope I can get your ear nonetheless.

You start the editorial with two questions: “Is looking like MS Windows good or bad?” and “Is KDE a better answer than GNOME?”

You spend half the editorial arriving at an answer to the first question with which I agree: “It depends.” For certainly some people, fresh in out of the wilderness, will like to have a Windows-like interface, for a while anyway. Whereas others will want to put childhood behind them. Fortunately everyone can have what suits them.

But of course it is the second question that is bothersome. It reminds me of a similar question: “Is emacs better than vi?” Haven't we all learned that this kind of question is silly and irritating?

Don't you value competition? don't you see the benefits that ensue? to name just one example: would Troll Technologies have modified their license for qt if they, and the KDE community, hadn't felt GNOME's breath at their backs?

To try to pick a “winner” between KDE and GNOME is like trying to pick a winner between emacs and vi. It is IMHO silly and, worse, unproductive. Let's have (at least) two wonderful editors, and let's have (at least) two wonderful windowing environments. That is the way for Linux and Open Source to make progress. That is what will benefit us consumers the most.

For the record, I am a charter subscriber to *LJ*, which I love, and: my window manager of choice is fvwm.

Best wishes,
—Alan McConnell, alan17@wizard.net

Learning Python review in last LJ

To: Phil Hughes

I just wante to write and let you know how much I agree with your review of Learning Python. I haven't actually read the book however, I just tried to learn python by it's predecessor. I'm glad you found the new book enjoyable and useful.

I started trying to learn Python from 'Programming Python' and quickly got bogged down. That was last Christmas. Recently, I picked the idea of python again, and came to a miraculously useful conclusion:

The OOP stuff doesn't matter until you want it.

When I started trying to write scripts as ordinary, straight through procedural stuff, just like my perl and ksh scripts, it just popped straight

out! I was converting my perl scripts over and they became instantly more readable! I am now far more impressed with Python than I ever was reading the 'PP' book, even though the stuff in there is very interesting. Combined with Python's large library of Very Useful Stuff, you can just get on with writing useful scripts, which I suspect is exactly Guido vanRossum had in mind.

Of course, once you've spent a long time with it, you start to get a better feel for the OOP stuff. Right now, I'm just going over the 'PP' book again, and it's making more sense this time around. But it's not the best of introductions to a very lovely and gentle language.

Anyhow, I'm glad to see somebody else enjoying Python for what it is.

Many thanks for *LJ*; I've been subscribing since issue 9 and look forward to every issue.
—Dominic Mitchell, dom@myrddin.demon.co.uk

Is KDE the Answer?

I'm not going to strain Phil's flame-proof suit :-)

Still, I would like to offer some thougths to his article in the October issue of *Linux Journal*. Well—is KDE the answer? Yes, it is. Is GNOME the answer?

sure. Is Python the answer? Of course! Is Perl the answer? How could you think otherwise. GCC? Egcs?. You name it.

I don't tend to see the diversity of solutions and approaches in the free software/open source world as a liability, but as an asset. A rather important one. In the metaphor of evolution (sometimes an overstrained metaphor, I admit), this diversity is the big genetic pool which ensures adaptability to changing environments. And environments do change in this field.

Besides, in this FS/OSS world, much of the development is fun-driven. The idea is that you are much more productive if you really enjoy what you are doing. There are those (I'm one of them) who shriek in horror at the idea of programming in C++, don't like particularly Python and are comfortable with the object models of Perl or Lisp. I'd rather go through the contortions of doing “more-or-less-kinda-OO” programming in C than “real-OO” programming in C++ for example. There are those who cry in disgust at this very perspective. A rich ground of tools and approaches will do much more to harness the energies of all of us.

I do hope that things get constantly reinvented to some extent. You can't afford to do radical changes to ext2, for example. It has to be a rock-solid basis for many production environments out there. But perspectives would be very dull without ext3 or reiserfs, for example.

The most important thing is to work on interoperability, and the Gnome and KDE folks seem to take this seriously (the last time I looked into their mailing lists, that is). It should be made as easy as possible to write applications which fit as well as possible into *both* environments. Then we get something the closed-software world hasn't: cross-fertilization. This is a concept which has been very successful in natural evolution—it might work for us as well. The easier it
is to “steal” software from each other the better.

So to offer my personal answer to your question: “Yes, definitely, all of them”. P.S. —again I enjoyed your magazine from cover to cover.

Thank you for the good work.
—tomas, tomas@aura.de (tomas zerolo)

Linux after CP/M

I am 46 years old and stopped programming when Big Blue came along. I used to work with the then available 8 - bit machines and hexadecimal coding. Now I decided to jump back into the programming area and started with the installation of RedHat 6.0.

I must say it is like being reincarnated for the world seems to be dominated by C++ and Internet. While instaling RedHat I noticed that one should have benefits of an education in LAN or some kind of Networks.

It also seems to me that the package will sell but that a lot of buyers will knock off. The road to completion is hazy for one will never be at ease and done. I live in the Netherlands, my e-mail adres is

groetjes,
—Robert Vencent Racam, evergladenld@planet.nl

Letter to the Editors

Two articles in October's issue of the *Linux Journal* concerned me a little. First of all, I have to find issue with the rather unfair article that Mr.Hughes' wrote in regards to the KDE/GNOME issue. First of all, QT is still not free. Nope. It's semi-free, it's sorta-free, but it remains a restrictive license, and the proliferation of these semi-free licences is potentially quite damaging.

We all appreciate the efforts Troll has made in order to loosen the license, but having the entire Linux desktop controlled by a for profit company is no different then Windows. If GNOME was the standard, we'd be able to have a wealth of applications, closed and open. However, with KDE the standard, commercial vendors will be less inclined to port if they have to fork over a large sum of money.

Mr.Hughes has condemned the GNOME project even though it has had far less time to reach stability. Recent releases, like the 1.0.53 builds have

shown remarkable progress and I use it daily, under fairly strenuous conditions. By his thinking, we should go with whatever works, even if it doesn't satisfy us. If Linux believe that, we'd all still be running Windows, and the guys at Red Hat would have far less money.

While it's not crucial that all the applications are free, it it crucial that the underlying system be free. QT is not truly free, while GTK and GNOME are.

On another note, I'd like to mention to Mr. Kroll that Code Warrior works well on other distributions. I'm a Stampede developer, and I use Code Warrior frequently. It was a simple matter of running alien, and I had a working copy of Code Warrior.
—Aubin Paul, outlyer@stampede.org

Article Idea

I think somebody should do a good article/tutorial on how to disable the many network services that users often don't need. I recently learned how to do this and by taking out nfs, mountd, sendmail, portmap, and probably others I can't recall, I have made my computer faster and more secure. My bootups are noticeably faster, the memory requirements of my system have been dropped by several Mb and I don't have to worry about security holes in all those daemons I don't understand.

I suspect that many people who, like me, use their linux boxes as workstations and not servers could benefit from an article telling them that this is OK to do. For people in my kind of situation, all that is really needed in terms of network servers is ftp and telnet. I've known for a long time that this is all I wanted to do, but I wasn't sure which daemons I could safely disable or the best way to do it, for that matter.

Just an idea,
—Colin Durocher, conan2@home.com

Someone has—look for it in our December issue. —Editor

Please Read: suggestions for LJ

Ms. Richardson, Mr. Searls,

I have been a subscriber to *Linux Journal* for a few months now, and a reader for a long time. I have some general comments to make with regards to the direction of the magazine that I hope you will consider. Let me say that I am only forwarding these suggestions to you because I would like to think I can hold your magazine to a higher standard than other linux magainzes that have jumped on the bandwagon.

Firstly, I would appreciate it if the magazine moved away from such a position of 'hollow' advocacy. Obviously I don't mean you should stop promoting linux - its your business, but it seems that the magazine is always treating me as an NT user that isn't convinced of the usefulness

of linux. This theme is so pervasive in your magazine that it even gives way to rather questionable quotes used to bolster your advocacy:

“...Since I have to work with NT for political reasons, I just cope with it. But I know if we could see the source, we could probably fix the problem pretty fast.” Windows NT is millions of lines long. Even if it was open source and well documented, it is *highly* unlikely that a single user, not deeply experienced with the architecture of the OS, could *ever* locate and repair the errors in question. Whether open sourcing NT would have prevented such bugs from appearing in the first place is another issue, but suffice to say, all of the open source operating systems have longstanding bugs, code availability nothwithstanding!

Secondly, I would like to see more meat in the interviews. This month's issue featured an interview with Linus Torvalds - a great opportunity to get the inside scoop on future developments, but instead we get a barrage of incredibly uninformative, almost insipid questions about his personal life. I know this last statement is rather rude sounding, but I cannot express to you how disappointed I was with the lack of content in this interview, which I was really looking forward to.

Thirdly, I would like to see more criticism of products and software. Some open source software is good. Some is terrible. I think users need to know about both kinds. As for hardware, I was disappointed that this month's review of VAResearch computers failed to mention cost as a criterion of purchase. VA boxes are historically overpriced by a significant margin, and your readers deserve to know that nearly identically equipped PCs can be had for much cheaper. I understand they advertise heavily in your magazine, but that should not preclude an honest appraisal of their products.

Other than that, “Best of User Support”, “New Products” and “At the Forge” continue to be excellent columns. I hope Mr. Lerner continues writing for your magazine for the forseeable future - his articles are extremely detailed and informative.

Thanks for your time
—Brad Clawsie, brad@yahoo-inc.com

“...Since I have to work with NT for political reasons, I just cope with it. But I know if we could see the source, we could probably fix the problem
pretty fast.”

Funny, I didn't see it as Linux advocacy, hollow or otherwise. I saw it as a way to show how the software industry is changing from one we understand in terms of its suppliers to one we understand in terms of its builders. The builder I quoted was remarking on the prefab nature of NT. The world of software, like the world of building construction, has plenty of room for all kinds of construction materials and techniques, including

the relatively prefabricated and closed offerings from Microsoft and other software suppliers.

But he didn't say that, and I didn't make it sufficiently clear. He did slam NT, however, with a statement you did well to criticize. And you're not the only one to make the same point. In fact, Craig Burton has been pounding on similar issues for some time. Witness this interview: http://www.linuxgazette.com/issue41/searls.html .

The other points you make are also helpful. And believe me: we do appreciate them. Regards,
—Doc Searls, doc@searls.com

Comment on "Microsoft Lamblast Linux"

In response to Microsoft release about 'Linux'.

We know that 'Salespeople' will spend more time lambasting a competitor with innuendoes instead of facts.

The best response, of a dignified nature, is to publish the results of comparisons from this months *Linux Journal*. This article list the test conditions and the results. You don't see Microsoft doing the same. Just muddy the waters with trash (verbal).

We in the Linux community should feel proud of the high priced advertisement and acknowledgment from such a large corporation.

I myself just laugh at the trash coming out of Redmond, but you don't see them doing a comparison of their system (Windows or NT) against Linux.

I believe, the best way to respond to this trash, is to not say a word (no flaming comments) and let the educated people of this world see how childish a large corporation can get. This corporation must think the people of the world are just a bunch of dumb people and cannot see through the trash.

Just think of all the free publicity Linux is getting. I say 'Thank you, Microsoft for your comments about Linux. Keep it up.' They are only making themselves look like a bunch of fools.
—Dr. Fred Lerssen, flerssen@worldnet.att.net

Hurt by Linus interview (Nov 99)

Marjorie;

I was hurt by your interview with Linus Torvalds in the Nov 99 issue of *Linux Journal*.

As linux user and person of faith, I am interested in Linus' spiritual health. I am not interested in having 4 questions, 3 with a decidedly anti-

religious slant, give air to the same tired excuses people use to not participate in corporate worship.

The article was to give the reader a picture “of who exactly Linus is”. But the effort changed mid-stride from a personal glimpse to slamming those who are faithful.

“Religion is a personal matter”. Linus was correct. Some time ago you set a journalistic standard to avoid certain uses of language because you felt it important to remain sensitive to other's beliefs. I ask that you continue that stand. Report and comment on Linux, and the state of open source software in general. Please avoid offending readers with comments that have nothing to do with the magazine's topic or scope.

Sincerely;
—Leam Hall, leamhall@rcn.com
An *LJ* reader since Aug 95.

Sorry, you were hurt. I didn't think we spent much time on religion and didn't consider it offensive. Even if it was the “same tired excuses”, it is what Linus beieves. I don't think either of us thought we were slamming the “faithful”. Why would I want to do that? I am one of the faithful.
—Marjorie Richardson

Complaint Department

I would like to register my complaint concerning your recent interview with Linus Torvalds. It seems to me there was an inordinate amount of space devoted to Marjorie's and Linus' bashing of “organized religion”.

As a journalist, I thought this showed questionable ethics and poor editing on your part (at times, it almost seemed as if Marjorie was egging him on).

As a Christian, I was offended that a magazine to which I subscribe for news and information relating to a computer operating system should devote so much space to bashing Catholicism, religious practices (such as tithing) and beliefs that I hold dear. So, Linus doesn't like organized religion. How about sticking to the topic in the future?
—Thomas Long
tlong@billsoft.com

Getting to know Linus and what his beliefs are in many areas *was* the topic.
—Editor

Archive Index Issue Table of Contents

Advanced search

UpFRONT

Doc Searls

Issue #68, December 1999

Stupid Programming Tricks, *LJ* Index and more.

LJ Index - December, 1999

1. Number of hosts on the Internet in December, 1969: **4**
2. Number of hosts on the Internet in August, 1981: **213**
3. Number of hosts on the Internet in October, 1989: **159,000**
4. Number of hosts on the Internet in January, 1992: **727,000**
5. Number of servers on the Web surveyed by Netcraft in September, 1999: **7,370,929**
6. Number of Apache servers on the Web: **4,078,326**
7. Apache's share of all web servers: **55.33%**
8. Apache sales: **$0 US**
9. Number of web pages with the term “brand”: **2,302,060**
10. Number of web pages with the term “branding”: **183,510**
11. Number of web pages with the term “brand name”: **114,262**
12. Money spent on advertising worldwide in 1998: **$200.3 billion US**
13. Money spent advertising Apache through all of time: **$0 US**
14. Number of web pages with the phrase “Apache”: **286,619**
15. Estimated consumer purchases over the Web in 1999: **$31 billion US**
16. Estimated business purchases over the Web in 1999: **$80.4 billion US**
17. Estimated annual consumer purchases over the Web by 2003: **$177.7 billion US**
18. Estimated annual business purchases over the Web by 2003: **$1.1 trillion US**

Sources

- 1 to 4 from Matrix Information and Directory Services
- 5 to 7 from Netcraft
- 9 to 11 and 14 from AltaVista, September 21, 1999
- 14 excludes pages with the word “native” or “Indian”
- 12 from McCann-Erickson
- 15 to 18 from International Data Corporation (IDC)

Deplugging the Net

> You almost certainly think of the Internet as an audience of some type—perhaps somewhat captive. If you actually had even the faintest glimmering of what reality on the Net is like, you'd realize that the real unit of currency isn't dollars, data or digicash. It is reputation and respect. Think about how that impacts your corporate strategy. Think about how you'd feel if a guy sat down at your lunch table one afternoon when you were interviewing an applicant for a vice-president's position and tried to sell the two of you a car and wouldn't go away. Believe it or not, what you want to do with the Internet is very similar. Just as you have a reasonable expectation of privacy and respect when you're at a table for two in a public place, so too do the users of the Internet have a reasonable expectation of privacy and respect. When you think of the Internet, don't think of Mack trucks full of widgets destined for distributorships, whizzing by countless billboards. Think of a table for two.
>
> —@Man, from “Attention Fat Corporate Bastards!” www2.ecst.csuchico.edu/~atman/attention-fat-bastards.html

Whoa, Nettie!

Alan Greenspan has stated that the Internet is the engine “driving” the U.S. economy. This engine has been working hard for a number of years, doubling and redoubling. But what happens when it begins to slow down?

This is not an idle question, as data in Matrix Maps Quarterly 601 (published by MIDS, http://www.mids.org/) demonstrates. The sky may not be falling, but growth is definitely slowing.

Comparing the resulting per-country host data for the period between January and July 1998 and 1999 reveals an interesting feature of the Internet. For nearly a decade, the number of Internet hosts doubled every year. From 1985 to 1997,

growth was 2.176. The rate of change is now 1.5. The graph outlines the growth rate for 1969-1999, then extrapolates out to 2002.

In some ways, the more things change, the more they stay the same. The total number of Internet hosts continues to grow, but at a somewhat slower pace.

RIPE data, published monthly, shows that the European growth rate has been falling off as well.

That this would happen should surprise no one; clearly, as any area begins to become saturated, growth slows.

—Peter H. Salus, info@linuxjournal.com

Overheard

> Do you want to spend the rest of your life selling sugared water, or do you want a chance to change the world?
>
> —Steve Jobs to John Sculley, 1983

> C'mon Steve, do you want to go on selling colored plastic all your life, or do you want to change the world?
>
> —USENET posting to Steve Jobs, 1999

jbum rocks

Jim Bumgarner's Public Opinion Research Project at www.jbum.com/jbum/public_opinion.html takes the Sucks/Rules-O-Meter concept to its extreme. His site lets you poll the Web (through www.altavista.com) for the negative and positive adjectives of your choice. Here are a few of the polls displayed at the site on September 23, 1999.

GAMES FOCUS—Weiqi Baduk Igo Go

a robot playing a board game or striking a thinking pose?

From its origins in China, Weiqi has spread through Asia and indeed throughout the world. In Korea, where it is most popular, it is known as Baduk, while in Japan and elsewhere, it has the familiar name Go. Its rules are simple and easily acquired, yet the mathematical and logical complexities are vast and largely unfathomable. Indeed, some say Go can be as demanding and subtle as art or science, while the scope for personal expression is such that it is said one

cannot hide his personality on the Go board. So what is your Linux box's personality like? Go find out!

The board is 19x19; the stones are round. Everything else is just numbers, a dendrite path which I would *vaguely* estimate to contain a number of paths somewhere around 361!/111! (if we assume 250 moves per game). Navigating this path well has proven to be an impossible task for computers. Still, in the post-Deep Blue world, Go may be both the most promising and the darkest frontier in artificial intelligence gaming. From chess, we developed heuristic searches and witnessed the power of brute force when we trimmed our search trees, not to mention the parallel processing research which occurred during the construction of Deep Blue. The subtlety of Go, which is less materially dependent than chess and has more branches in the dendrite with a less clear objective (territorial acquisition rather than monarch hunting), has rendered brute force largely ineffectual, and we have to resort to pattern recognition and analysis and strange algorithms to make a computer think like a human. What we learn from developing Go software could expand our knowledge base of AI theory and techniques a great deal. The computer chess authority Hans Berliner remarked that Go “may have to replace chess as the task par excellence for AI.” Fortunately for us Linux enthusiasts, there is some very high-quality Go software floating about, free of charge and with the source code, of course.

If you have long been frustrated by GNU Chess, you can get back at the GNU Project by beating GNU Go—well, maybe not. While computers are relatively worse at Go than at chess, GNU Go is not a weak program. In fact, it recently took second place at the 1999 U.S. Computer Go Championship, winning the “Best New Program” award as well. It has a text interface, but since it understands Go modem protocol, it can also be played against other programs or with the Cgoban interface. Check out www.gnu.org/software/gnugo/devel.html if you're interested in playing or contributing to the project.

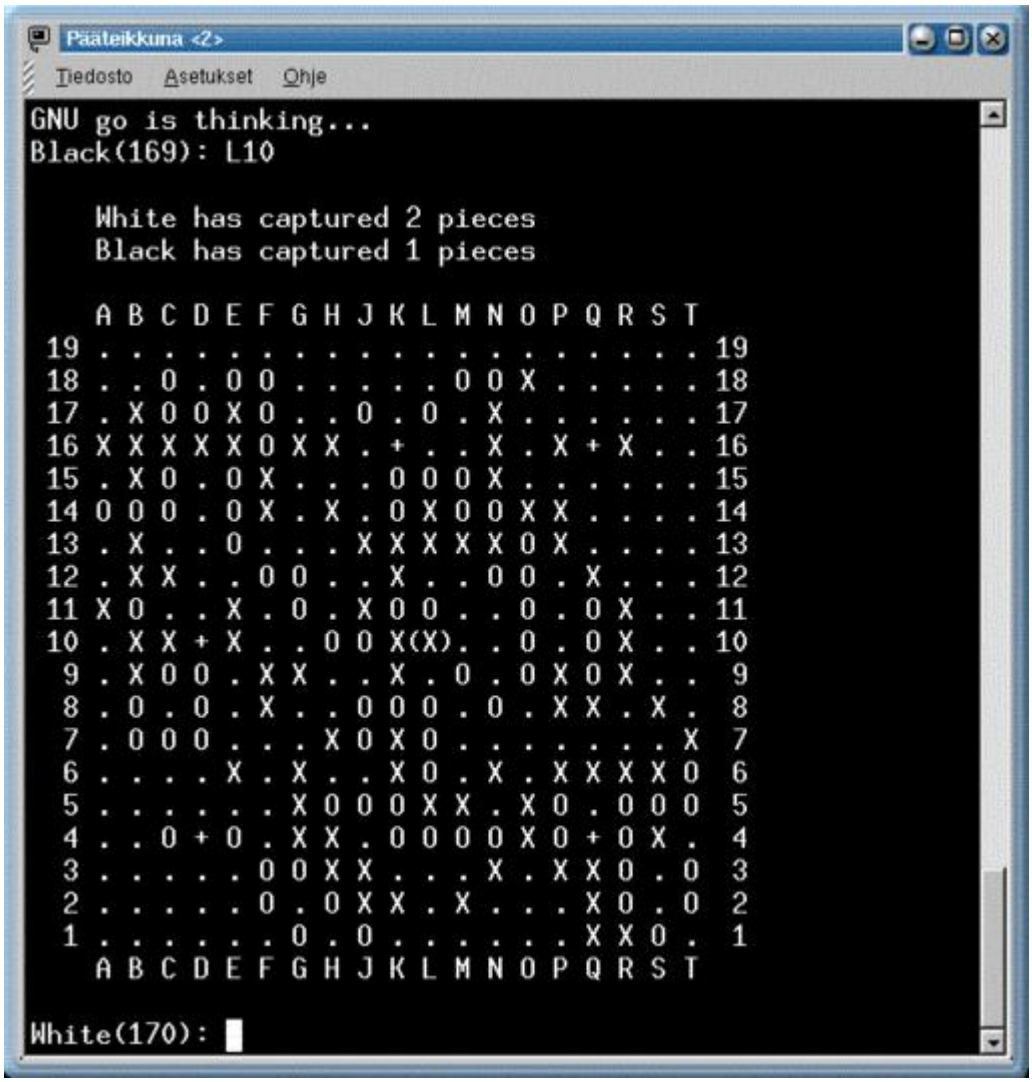


Figure 1. GNU Go for Console Enthusiasts

Baduki, by Jim Laebum (who goes by Artist), is free Go software with its own graphical interface. The program is actually quite good, and the interface is nice. The board seems to have come from the GIMP's wood pattern (I suspect, since I did this myself for Go software I was writing). The software allows you to set handicaps and play levels, and the interface scales to whatever window size you like. Additionally, Baduki can give its rationale for moves, show its thinking process and display alternate moves. Baduki understands GMP (Go modem protocol); thus it can play on IGS (Internet Go Server), NNGS (No Name Go Server) or against another program such as GNU Go. The Baduki home page lives at soback.kornet21.net/~artist/baduk/baduki.html.

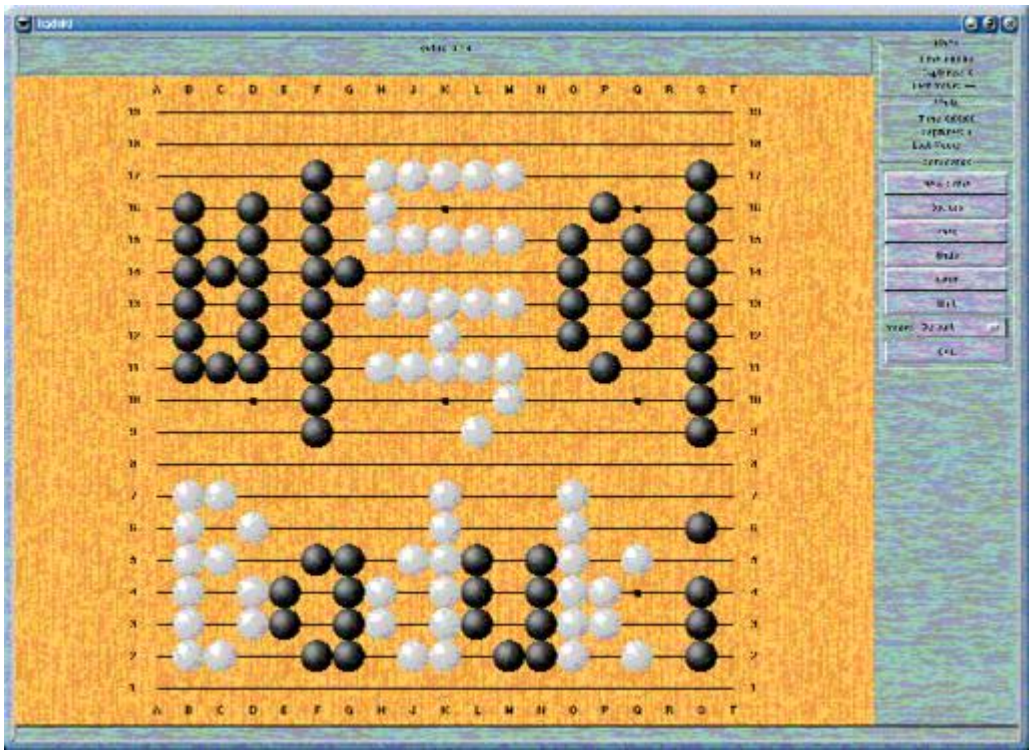


Figure 2. Jim Laebum's Baduki

CGoban (Complete Goban) by William Shubert is the Go equivalent of xboard. It allows users to play Go against programs (such as GNU Go or Baduki), or against other players on the Internet Go servers. In addition, CGoban allows one to examine SGF files, that is, game records. The interface looks quite nice (the standard wooden board with black and white stones), scales to whatever size you like, and should run on all UNIX systems with X. CGoban will automatically connect you to the Go server of your choice with a simple mouse click. The home page of CGoban is http://www.inetarena.com/~wms/comp/cgoban/.

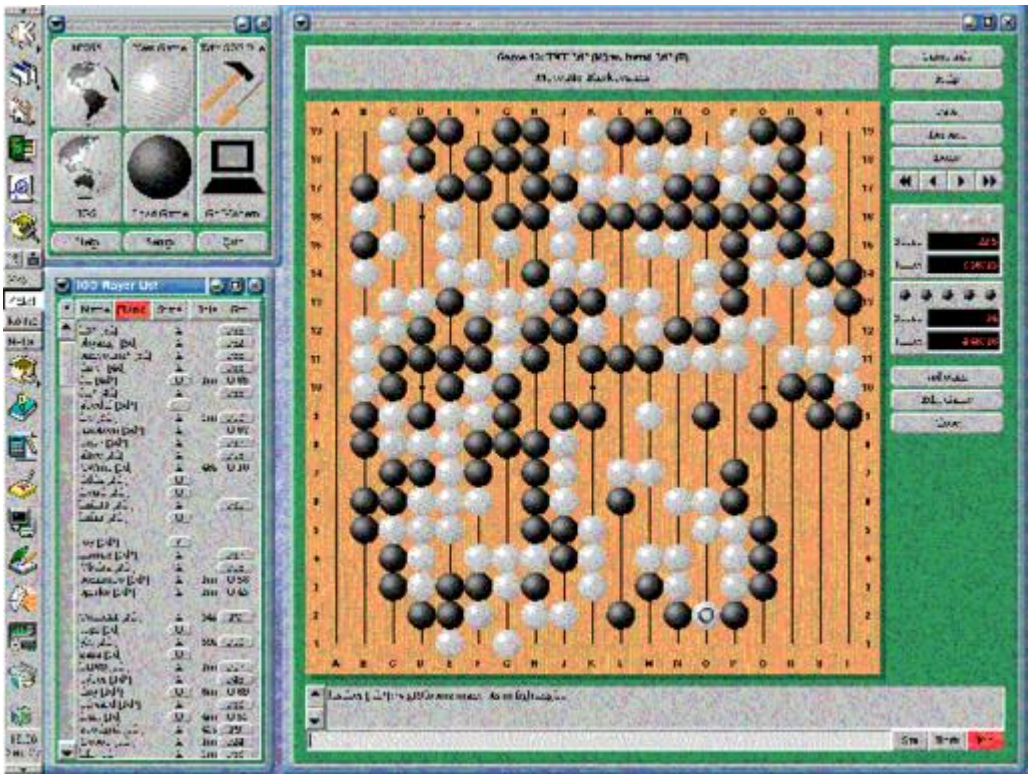


Figure 3. CGoban on the Internet Go Server

The Internet Go Server is the on-line meeting place for wired Go enthusiasts the world over. You can **telnet** in and play with a text interface, or use a graphical interface like CGoban. IGS is similar to the chess servers, with the typical commands applying like who, match, observe, kibitz, tell and shout. I find the atmosphere to be friendly enough, though less talkative than the chess counterparts. Also, blitz Go seems to be less popular than blitz chess, and as for lightning Go, I don't know. (Lightning chess is one or two minutes per game, a rather difficult schedule for Go.) If you want to check out IGS's well-designed web page (available in English, Chinese, Korean and Japanese), go to http://igs.joyjoy.net/. Or, if you want to go directly to the server, use **telnet igs.joyjoy.net 6969** (yes, port 6969). There are a few other Go servers wandering about, No Name Go Server (NNGS) being one of the more popular. CGoban already knows the addresses and will connect you automatically.

Whether Go interests you on a playing level or a programming level, many on-line resources are available as well as software and many excellent books. A visit to your local bookstore or gaming shop should provide you with ample opportunity to foster an obsession (assuming, of course, that you don't find it boring). Likewise, if AI is your thing, Go is in need of creative solutions and has a lot of scope for truly clever, brilliant thinkers. If you become outstandingly fond of Go, you may even want to check out your local club, which would probably be overjoyed to have a new member. Happy Going! (Next month, something more violent...)

—Jason Kroll

STUPID PROGRAMMING TRICKS→SCROLLTEXTS

Last episode, we initialized console graphics (at great risk to our system's health) but we didn't do much afterwards. Now it's time for something truly impressive, a feat which Microsoft has apparently not yet accomplished—a smooth scrolltext.

“What is a scrolltext?” you may ask, in particular if you haven't been around on a Commodore 64, Amiga or PC and names like Fairlight, Red Sector and Future Crew mean very little to you. Hopefully, you've at least seen the intro screen from Jet Set Willy. What it comes down to is this: scrolltexts are the most exciting form of communication ever. Words that glide across the screen, often advertising the latest release or copy party, complaining about high school teachers, or detailing unfortunate experiences at the hands of public transit, usually accompanied by music (MODs) blaring in the background with animated graphics and the ubiquitous star fields. Perhaps you've seen Microsoft's screen saver which scrolls words across the screen and noticed it flickers terribly. It is *very* simple to fix, so our scrolltext won't flicker; then we can taunt the Microsofties to fix their screen saver.

The routine is quite simple. We start by initializing three graphics screens: a physical screen, a virtual screen and a scroll board screen. The physical screen is the graphics context which will be displayed, the virtual screen is the graphics context we use as a whole-screen copy to the physical screen. That is, we make changes to the virtual screen, and when everything's ready, copy it to the physical screen. The third screen is the scroll board, a virtual graphics context which will be wider than the physical screen by one character (8 pixels) and will be only as tall as the font itself (again, 8 pixels). Since we're using an 8x8 font and a 320x200 graphics screen, we can fit 40 (320/8) characters on a line, making 41 characters for our scroll board. Once we've set up these three contexts, we'll use a simple loop to get the letters scrolling. After that, we can add anything we like: 3-D graphics, dancing animals, star fields or anything else. Here's the loop:

- Write 41 letters to **scroll_board**.
- Copy 40 letters from **scroll_board** to **virtual_screen**, always copying to the same location on **virtual_screen**, but copying from one pixel farther to the right each time, so that at first we get the first 40 letters, then the first 40 letters minus the first row of pixels from the first letter but with the first row of pixels of the 41st letter and so on, until we have scrolled 8 pixels (the width of our current font).
- After each copy of **scroll_board** to **virtual_screen**, copy **virtual_screen** to **physical_screen** and hold for a vertical refresh. (This makes things look smooth and lets us add things later, like the dancing animals.)

- Once we have moved 8 pixels forward in **scroll_board**, such that we are copying the 2nd through the 41st letter (instead of the 1st through 40th), we reprint 41 letters to **scroll_board**, starting one letter in, such that what was once, for example, “Hello world welcome to my glorious scroll” would become, “ello world welcome to my glorious scrollt”.

This routine is fairly simple and requires only a couple of variables: one to keep track of how many pixels in we are and one to keep track of how many letters in we are. Also, we want to make sure we don't run out of scrolltext and start scrolling bits of random memory, which would ultimately lead to a segmentation fault. While we might prefer to draw each letter individually and just keep modulating around the length of the scrolltext, we get faster drawing if we print characters as a string once instead of calling **gl_writen** 41 times every time we move 8 pixels. So, leave some blank space at the beginning and end of your text to ensure smooth wrapping. We could also create the whole scrolltext as one really long graphic, but that would be cheating.

Once the basic scrolltext is going, we can do all sorts of fun things. We could, for example, have a sinusoid equation for the *y* value of where the text is placed such that it would bounce up and down on the screen, or we could insert some graphics and possibly call **mikmod** or **playmidi** to get some music going. Compile with

```
gcc -Wall -O2 scrolltext.c -lvgagl\
   -lvga -o scrolltext
```

For information on the specifics of **svgalib** or **vgagl**, try their respective man pages. Library functions also have their own man pages. Here's the code:

```
#include <stdio.h>
#include <stdlib.h>
#include <vga.h>
#include <vgagl.h>
#define VGAMODE G320x200x256
#define FONTW 8 // font width
#define FONTH 8 // font height
#define TEXTL 600 // text length
int main(void)
{
  char d;
  char text[TEXTL]="                                        Megagreetings from whomever this happens
  short int text_pos;
  unsigned char pixel_pos;
  unsigned char speed;
  GraphicsContext *physical_screen;
  GraphicsContext *virtual_screen;
  GraphicsContext *scroll_board;
  vga_init();
  vga_setmode(VGAMODE);
  gl_setcontextvga(VGAMODE);
  physical_screen = gl_allocatecontext();
  gl_getcontext(physical_screen);
  gl_setcontextvgavirtual(VGAMODE);
  virtual_screen = gl_allocatecontext();
  gl_getcontext(virtual_screen);
  gl_clearscreen(0);
  scroll_board = malloc( (WIDTH/FONTW+1) * FONTW *
     FONTH * BYTESPERPIXEL);
```

```
    gl_setcontextvirtual(WIDTH+FONTW, FONTH,
       BYTESPERPIXEL, 8, scroll_board);
    scroll_board = gl_allocatecontext();
    gl_getcontext(scroll_board);
    gl_clearscreen(0);
    gl_setwritemode(FONT_COMPRESSED);
    gl_setfont(8, 8, gl_font8x8);
    gl_setpalettecolor(1,63,63,63);
    text_pos = 0; // text offset
    pixel_pos = 0; // pixel offset
    speed = 1; // scroller speed
    gl_setcontext(virtual_screen);
    for (d=0; d==0; d=vga_getkey()) {
      pixel_pos+=speed;
      while (pixel_pos > FONTW) {
        gl_setcontext(scroll_board);
        gl_writen(0, 0, WIDTH/FONTW,
          &text[text_pos]);
        text_pos++;
        pixel_pos-=FONTW;
        if (text_pos > TEXTL - WIDTH/FONTW)
        text_pos -= (TEXTL-WIDTH/FONTW);
        gl_setcontext(virtual_screen);
      }
      gl_copyboxfromcontext(scroll_board, pixel_pos,
         0, WIDTH, FONTH, 0, HEIGHT-FONTH-1);
      gl_copyscreen(physical_screen);
      vga_waitretrace();
    }
    return 0;
  }
```

This code can be downloaded from ftp.linuxjournal.com/ftp/pub/lj/listings/issue68/3722.tgz.

—Jason Kroll

VENDOR NEWS

Red Hat, Inc., a provider of open-source Linux-based operating system solutions, announced that **Burlington Coat Factory Warehouse Corporation** has purchased support services from Red Hat for its nationwide Linux deployment. Under the agreement, Red Hat Services will provide telephone-based support to more than 260 Burlington Coat Factory stores nationwide (including subsidiaries). Red Hat will provide ongoing maintenance for customized Dell OptiPlex PCs and PowerEdge servers running factory-installed Red Hat Linux. The Red Hat Linux OS-based systems will host Burlington Coat Factory's Gift Registry and will facilitate all other in-store functions, such as inventory control and receiving.

International Data Corp. research states that Linux was the fastest-growing server operating environment in 1998, growing more than 212 percent in that year alone and capturing more than 17 percent of new licensed shipments of server operating systems.

Andover.Net announced it has completed extensive hardware upgrades to its influential Slashdot (www.slashdot.org) and Freshmeat (www.freshmeat.net) news and resource sites. With a major investment in the IT infrastructure of

both sites, Slashdot and Freshmeat can now serve the growing Linux community, without delay, the same news and information that has made these sites the most popular destination for Linux news and information.

Oracle Corp., a provider of Linux-based database software, and Red Hat jointly announced that Oracle has certified Oracle8i on Red Hat Linux and that future Oracle product releases will also be certified on Red Hat as they become available.

Tripwire joined ISS' Adaptive Network Security Alliance (ANSA), an industry-wide initiative dedicated to developing and delivering adaptive network security solutions. Through ANSA, real-time adaptive security capabilities are being integrated across systems and applications, providing automated responses to security risks, including intrusions. Tripwire will use the ANSA modules to integrate Tripwire and ISS products. Any modifications made to operating system or user files will be detected by Tripwire. Tripwire will then send an alert to the ISS product set, which will conduct another set of security checks to monitor and combat the intrusion. More information on ANSA can be found at http://www.ansa.iss.net/.

KeyLabs Inc., an e-business testing facility, announced that Motorola Computer Group's (MCG's) SLX2020 network appliance has passed KeyLabs' network server compatibility tests in support of major Linux operating system distributions. KeyLabs' compatibility testing showed the SLX2020, the first of Motorola's recently announced SLX Series of network appliances, to be compatible with Caldera Systems' OpenLinux 2.2 and 2.3, Red Hat 6.0, SuSE Linux 6.0 and TurboLinux 3.0.1. KeyLabs' “Linux-Tested” certification results for the SLX Series may be found at www.keylabs.com/linux/results/motorola.html.

Ariel Corporation, a supplier of open-architecture remote-access solutions for Internet service providers, announced that **KeyLink Systems**, a Pioneer-Standard Electronics, Inc. company, is now offering bundled remote access solutions for ISPs based on Ariel's PowerPOP architecture. KeyLink Systems also announced a dedicated sales and support team for the assembly of these ISP solutions.

STRICTLY ON-LINE

Transparent Firewalling by Federico and Christian Pellegrin presents the solution to one of the difficult problems encountered when building a firewall: how to split the existing network without affecting the configuration of the machines already in use on the network. They do this by using a proxy arp technique. All the information you need to know about requirements and configuration can be found here.

Kerberos by Cosimo Leipold is an introduction to this powerful set of programs which give you encrypted connections to TELNET, FTP, e-mail, etc. Mr. Leipold explains the configuration files and commands needed to give the administrator complete control of the system.

What Can You Expect? by Denny Fox describes the end-to-end process of defining and implementing a data collection project that illustrates the use of Expect, stty, cron, a little C programming, gnuplot and ioctl to the serial device driver. Learn more about Expect, a powerful tool used to automate UNIX programs which interact with a user or processes needing a command or trigger and then return some kind of response. Just the sort of tool System Administrators need on a regular basis.

Building a Firewall with IP Chains by Pedro Bueno is a very short article which gives you the basics on using IP firewall chains. Developing adequate security for your system is one of the most important steps you can take.

Customizing the XDM Login Screen by Brian Lane shows you how to jazz up your login screen, explaining how to set up XDM, change your background pattern, randomly display a background image and change your prompt.

The Use of Linux in an Embedded System by Dave Pfaltzgraff presents one company's solution to a customer problem using Linux and open-source software. Mr. Pfaltzgraff tells us how to implement the serial interface and control program and interface with the database, in his case, PostgreSQL.

Porting Progress Applications to Linux by Thomas Barringer is an explanation of the steps required to port an existing Progress application to the Linux system, including the advantages and disadvantages of doing so.

Army National Guard Using Linuxby Richard Ridgeway is a look at how a military war game simulation was ported to Linux workstations. Included is a comparison of graphic refresh times on different platforms and operating systems. Saving money and getting high performance are two very good reasons to port to Linux.

COSOURCE ANNOUNCES COMPLETION OF FIRST PROJECT

The program **supermount** has been successfully ported to the 2.2 kernel, implementing all the functionality of Stephen Tweedie's original version. The project was started on August 11 and completed on September 27 by developer Alex ... in Russia. This Linux enhancement was cooperatively funded by several different individuals and coordinated by cosource.com, thereby proving the Cosource model works! Congratulations to Alex and Cosource! Get

all the details from www.cosource.com/cgi-bin/cos.pl/bid/info/5http://www.cosource.com/cgi-bin/cos.pl/bid/info/5.

Lineo Proposes Embedded Linux Advisory Board

On September 30 at the Embedded Systems Conference in San Jose, Lyle Ball and Bryan Sparks of Lineo proposed a group to be called the Embedded Linux Advisory Board (EMLAB). This proposed body would serve as an advocacy group, helping Linux to gain greater visibility and name recognition in the embedded systems arena through activities such as establishing Linux Pavilions at embedded systems shows and promoting birds-of-feather sessions and Linux presentation tracks. Other possibilities include:

- Shared software development, for example, a GPL flash-disk file-system driver. Like Linux itself, such software offers a fundamental basis on which embedded systems may grow.
- Vendor-neutral comparisons of embedded Linux approaches
- Tracking and publicizing Linux design wins
- Setting standards

Funding would come from corporate sponsorship with complimentary memberships open to community groups, such as Linux Router Project or individual developers, via a nomination process.

Lineo hopes to turn EMLAB over to an independent board to be selected soon.

Present at the announcement were Lineo and some of its customers and strategic partners including Ziatech, Motorola and Intel. The press was represented by *Linux Weekly News* and *Linux Journal*.

Reaction from companies and groups not present was cautious, though there is support for the idea of such an organization.

Lineo has set up a server hosting an open mailing list and a web site. For list subscription information, e-mail info@emlab.org. For news updates, visit http://www.emlab.org/ or stay tuned to www.linuxjournal.com/.

—Dan Wilder

Archive Index Issue Table of Contents

Advanced search

Millennial Musings—Y2K

Peter Salus

Issue #68, December 1999

The whole to-do seems both parochial and silly.

There are many folks who worry about planes falling out of the sky and civilization ending at 00:00:01 on January 1, 2000.

Of course, this is the date only for Western Christians. For Eastern Christians, the calendar is shifted by about two weeks. For Jews, it's about 240 years until the (next) millennium. For Moslems, it's also several centuries. So, the whole to-do seems both parochial and silly. Also, for computer folk, there's the realization that the calendar isn't a UNIX system: it doesn't begin with “0”, so the second millennium must begin on January 1, 2001. (If you think there was a year 0 between 1 BC and 1 AD, this will be where you stop reading.)

However, there *are* problems, and they are worldwide. Why? Because over the past 50-odd years, chips and computers have become ubiquitous. Computers don't actually think very well. Unlike bookkeepers from the Italian Renaissance to the present, who used two-digit dates and had no problem computing interest for 1695 to 1705 when they wrote “95-05”, computers don't really know how to handle “00.”

So what?

Well, in most cases we just don't know. It appears that the overwhelming portion of the U.S. power grid has tested okay. But what if a very small supplier “trips” something? The grid folks assured those of us living in Manhattan in 1965 that a massive grid failure couldn't happen again. There was another less than a decade later. A few years ago, there was a cascading failure on the West Coast. So we really aren't sure what will happen.

On the other hand, there are myriad things each of us can test: kitchen and household appliances with date chips, for example. I tested my VCRs, setting

the date to December 31, 1999. In the morning, one had the right time, one was blinking “00:00” at me.

Maybe my coffee maker won't go on, or my bread machine, or the VCR won't tape the *n*th episode of something. But none of these is necessary to my life.

I have confidence in the various phone companies, too. They have far too much at risk not to have checked things out. And I know that most electronic switching systems are Y2K safe.

Doors with time locks may not be. I don't know whether banks have checked the locks on their doors and their safes as rigorously as they have checked their accounting machinery. But I know the New York Stock Exchange is in good shape, as are the large brokerage houses.

What About Computers?

If you're running UNIX, time doesn't run out for decades; well, at least into the 21st century for Linux. DOS 6 and earlier? Win95? You may be a victim. Windows 98? Well, according to *Agence France Presse* last February, the French Directorate for Competition and Prevention of Fraud (DGCCRF) showed that Windows 98 and Works 4.5 would be unable to recognize the year 2000. According to Marylise Lebranchu, the Minister for Small and Medium Businesses:

> The DGCCRF carried out tests in mid-January on products which might not work after 2000 and we have proof that Works 4.5 and Windows 98 will not work. It is extraordinary that a company which is supposedly at the cutting edge of technology has sold products which will not work after 2000.

And the Internet?

One of the design basics in the ARPANET, the sire of the Internet, was that it be both redundant and resilient. Today's Net, with over 50 million host machines and about 150 million users, is even more redundant. Many of us “live” on a frail branch, a mere twig, a single leaf. We depend upon an ISP—an enormous one like UUNET or a small one with perhaps 100 to 200 clients. Many of these smaller ISPs run older routers.

Go look at Cisco's site. All the routers currently sold are Y2K sound. Go back a couple of years: either all are sound, or there is free software available for download over the Net. If you go back five years, not only are the routers not Y2K-safe, they don't have enough memory to be upgraded.

So I worry that Internet users at some small twigs may lose their dial-up connections. I worry that some DNS caches may fail or be corrupted. Most of all, I worry about people's reactions to not being able to dial up, not being able to see the ball drop in Times Square from their desktops (as opposed to their TV sets), or not being able to exchange chat New Year's greetings.

If one looks at the Net statistics for hurricane Floyd last September, it's clear the usage spikes came after the hurricane passed: folks wanted to know “How bad was it?”, “How's uncle Fred?”, “Are you okay?”

So the big Y2K events may occur before noon in the continental U.S. on December 31. When it's midnight in Guam, it's 8 AM in New York and 5 AM in Silicon Valley. In the US, we'll have a lot of warning: from Tokyo, Melbourne, Singapore, etc., as the hour marches around the globe.

I don't think we should worry. There may be glitches and minor failures, but it won't be the end of the world. Penguins will survive.

Peter H. Salus, the author of *A Quarter Century of UNIX* and *Casting the Net*, is Editorial Director of SSC.

Archive Index Issue Table of Contents

Advanced search

Tale of Two Markets

Doc Searls

Issue #68, December 1999

“Free markets are self-organizing, permitting the most efficient use of resources for the greatest creation of value.” —Adam Smith

“Each year when it comes time to award the Nobel prize in economics, it goes to the economist who most eloquently paraphrases Adam Smith.” —Michael Tiemann”

Let's start by savoring the irony that the biggest thing ever to hit business—the Internet—was the creation of hackers working with government money. They didn't do it for the money, and they didn't do it to make money. They did it because it needed to be done and doing it brought a priceless kind of esteem: the respect of their peers. Crafting the Net was a good and noble work that still goes largely uncredited outside the circle of peers who best understand what actually happened. Given the importance of that work to the economy, it's a wonder these guys don't have statues on Wall Street.

Now the stock market is going gaga over Linux for the same reason it went gaga over the Net: it's good for business. *Why* doesn't matter to the market. Putting Linux to work is kind of like putting an *e* in front of your idea or a .com after your company name. It's a grace. A particularly lucrative grace, it now appears.

But just as the Big Boys didn't understand at first what that .com truly meant, they don't understand what Linux and open source mean, either. They thought the Net was just a 10-million-channel TV with a keyboard instead of a remote control. Likewise, they think Linux is just a 10-million-hacker version of Microsoft with bloated attitude instead of bloated software.

Not that the “free software” community is especially understanding about Business. Scott Lanning had this to say about *Linux Journal* for a *Seattle Times* story: “It's proprietary, so it means nothing.”

Makes me wonder: how are we “proprietary”? In *Open Sources: Voices from the Open Source Revolution*, Richard Stallman writes, “The `Linux' magazines ... are filled with advertisements for proprietary software that works with GNU/Linux. When the next Motif or Qt appears, will magazines warn programmers to stay away from it, or will they run ads for it?”

Well, probably both. Why shouldn't we give Troll Tech a way to say good things about Qt tools? What's wrong with that?

Answering this question will, of course, bring up blood-boiling moral questions which find their most concrete expressions in licenses and copyrights (also copylefts) that are strange and baffling to those who never heard of “open source” or “free” software until confronted with the need to use it. What's so “free” about something that comes with so many restrictions—and with so many people who are quick to flame you for breaking rules only they understand?

Business has a real demand for what Eric Raymond calls “software that doesn't suck”, and for the people who make it. Likewise, those people have a real demand for good development tools. And not all of those tools are going to be free or open source. Qt is an old example. The new example comes from Inprise (formerly Borland, also known as Borland/Inprise).

This past July, Inprise ran a survey of developers visiting various Linux sites including *Linux Journal*. Of the 24,000 unique responses, about 6,000 called Linux their primary development platform. Naturally, the leading development tools for this group were gcc/egcs (47%) and Emacs with gcc/egcs (27%). Yet a majority of these same 6,000 developers were willing to pay for commercial Linux development tools: 27.9% said they would be willing to pay $300 or less, and another 37.2% said they would be willing to pay $100 or less. Only 21% said, “Nothing, it must be free.”

What kind of tools did they want to buy? By class, the favorites were “Rapid Application Development IDE (RAD, visual development)” at 53.6%, and “traditional development IDE (integrated editor, compiler, debugger)” at 35.7%. As for particular tools, first choices were “C++Builder (C/C++ with RAD)” with 29.6%, “New IDE that works with existing standard Linux tools” with 24.2%, Delphi with 19.2%, and “Borland C++ (C/C++ without RAD)” with 13.3%.

Inprise has responded to demand with Kylix, described by Michael Swindell, who heads the project for the company:

> Kylix is the code name for a high-performance native Linux development environment—a Rapid Application Development environment—that will support C++ and

> Delphi development. It's a major effort. We're developing a visual component framework to radically speed and simplify native Linux development. Graphical, database, GUI, Internet, multi-tier development will all be completely visual and component-based, using the Borland VCL and two-way tool technologies. Just like Delphi and C++Builder for Windows. Its architecture is a component abstraction directly over the native environment. It is derived from the VCL architecture the current Delphi and C++Builder products are based on, so the component APIs and methodologies will be very similar. So familiar that if you already develop with Delphi or C++Builder for Windows, it will be very easy to do Linux development from day one. Essentially, you won't have to learn the complex details of the underlying architectures, though they'll still be completely accessible, making new Linux development much faster and porting far easier.

Now here's the really tricky part, because it's cross-ethos as well as cross-platform:

> We are not developing Kylix as an open-source project, but we are investigating which parts of Kylix lend themselves to being open source. Right now, Kylix will allow developers to create open-source applications. But we have not determined which, if any, components in Kylix will be open source. Our goal is to enable Kylix to develop both proprietary and open-source applications, because there will be markets for both.

As I write this (late September), Corel is getting beat up for putting a proprietary boilerplate on the beta version of its new Linux distribution. No doubt Inprise will run through a similar gauntlet, all the way to the market. In fact, Swindell expects it:

> This is a discovery stage that a new market has to go through. You've got commercial vendors coming into the open-source space, and open-source vendors moving into the commercial space. It's a merging of very different business practices. We will have to go through a lot of head scratching before this settles out.

How will each model subvert the other? Consider the parting words from Michael Swindell:

> We need the extremes. These are the people fighting for the noble causes. We can't discover open-source standards and policies without them. And frankly, open-sourcing everything is not a closed question for us. We'd welcome the Open Source community's involvement in our own development. But we've got

> fourteen years of intellectual property and patent accumulation, plus legal and shareholder interests to protect. There are extremes on that side, too. The best we can do right now is answer the market and talk to everybody with an interest in making this work.

The default assumption is that the most powerful interest—shareholders—will want Inprise's source code to stay closed. But it's not hard to imagine investor pressure on traditional software companies to free their source code, just as the same investors pressured .com companies to free their content.

Freedom is an efficiency that drives value. Isn't it fun to watch this new software business teach Adam Smith's lesson, one more time?

Doc Searls is the Senior Editor for *Linux Journal*. He can be reached at info@linuxjournal.com.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #68, December 1999

Our experts answer your technical questions.

Partitioning

I am installing and have a 3GB hard drive. I am not sure of the best way to partition it. Any suggestions? —Jes, jes2@mindspring.com

You do not say what you plan to do with the disk, or what the host machine is (server? workstation?). Personally, I usually partition as such:

```
/       (about 50Meg)
/safe   (same size)
/var    (half the space left)
/usr    (the other half)
```

I then link /home to /var/home and /tmp to /var/tmp/tmp. /safe is a copy of my root partition so that I can boot on it with LILO if my boot partition becomes badly damaged, and it also gives me a copy of my configuration files in /etc.

Adding more partitions can be nice, but the more you add, the more chances you have to run out of space in one of them. For instance, I think making /tmp a separate partition instead of linking it to /var is a stupid idea, unless you're willing to give it several hundred megabytes, or even a gigabyte, because some programs can create huge temporary files and may fail in bizarre ways if your /tmp partition is too small.

/usr is also meant to be read-only, which works very well on Debian and works mostly on Red Hat (they have incorporated most of my bug reports). In both cases, you'll still have to remount the partition as read/write before installing a package, however. —Marc Merlin, merlin@varesearch.com

Networking Oddities

I have tried to install Red Hat 6.0 twice now from the CD included in Sams' *Red Hat Linux 6 Unleashed*, and both times have run into the same problem. Installation goes fine, and when done, I can log in at the console with no problems, I can telnet out, ping, etc., and I can ping it from other machines on my internal network. When I use **telnet** to log in to the machine, I get "telnet: unable to connect to remote host: Connection Refused". When I try to do the **finger** command, I get the same error. When I use FTP to get to the machine from a 95 workstation, I get "Refused". When I try to do the **finger** command, I get the same error. When I use FTP to get to the machine from a 95 workstation, I get "ftp:connect :10061", and of course, it does not connect. On this machine, the IP address is 189.0.1.50, and the subnet address is 255.255.0.0. Interestingly though, Apache does work, and I was able to connect the Samba client to the Samba server, so TCP/IP seems to be working, but only some services.

My inetd.conf file has the lines

```
ftp stream tcp nowait root /usr/sbin/tcpd in.ftpd\
   -l -a
telnet stream tcp nowait root /usr/sbin/tcpd\
   in.telnetd
```

Further down, I also uncommented the line

```
finger stream tcp nowait root /usr/sbin/tcpd\
   in.fingerd
```

I have used Red Hat and other Linux flavors for over two years, and have never had a problem like this before. —Charles Almond, charles@ovis.net

I answered a similar question for a local user who had exactly the same problem just last week. Make sure your **inetd** daemon is actually running. All the file changes in the world do nothing if the daemon isn't present to read it. You can do a **ps ax** and scroll through the list of running processes to make sure it is present. —Chad Robinson, chadr@brt.com

There is another possible reason: the connections are being denied by tcpwrappers. Check /etc/hosts.allow, /etc/hosts.deny and /var/log/syslog. —Marc Merlin, merlin@varesearch.com

PPP Locked by Process

I'm trying to connect to my service provider. I got them to tell me how to set it up according to all their specifications through **linuxconf**, and according to them, the setup is right. When I run **ifup ppp0**, I get the following message:

```
pppd 2.3.7 started by root uid0 device is locked by pid438 exit.
```

How do I unlock ttyS1? —Kirk, webmaster@dcas.net

Looks like there is or was another previous process (438) running, which has or had the /dev/ttyS1 device locked. I think this is the port where your modem is connected. Log in as root, check if the 438 process is still running with **ps aux| grep 438**; if so, kill or terminate it with **kill -HUP 438** and check on the /var/lock directory for any file with a name like "LCK..ttyS1" which is the actual lock file. Remove it and try again. —Felipe E. Barousse, fbarousse@piensa.com

There is also the possibility that two programs are trying to lock the device when you try to dial out. If that appears to be happening (which is most likely the case if there is no lock file but pppd always dies), try removing any lines that say "lock" in /etc/ppp/options. —Steven Pritchard, steve@silug.org

RPMs, Downloading

I am an end user with a Caldera 2.2 Linux system. My question regards the installation of new software in the RPM format and the dreaded failed dependences with respect to a missing or wrong version of a library. I generally use the KDE **kpackage** program but have had the same problem with the command line **rpm**. Let me give a real example:

```
Download program.rpm
Run kpackage—reports: Unsatisfied dependencies
libgdk-1.2.so.0
libgtk-1.2.so.0
```

But, the Caldera 2.2 system has

```
libgdk.so.1.0.5
libgtk.so.1.0.5
```

Obviously, I need to install an update to GTK+ library. So, I downloaded gtk+1.2.3 which provides:

```
libgdk-1.2.so.0
libgtk-1.2.so.0
```

Next, I download the package and use kpackage to install. Guess what: **Unsatisfied dependencies** appears for this package:

```
libc.so.6 (glibc2.1)
libc.so.6 (glibc2.0)
```

How can two versions of this library be required?

```
libm.so.6 (glibc2.1)
```

Caldera 2.2 has GLIBC 2.1-3 which provides

```
libc.so.6
libm.so.6
```

How do you resolve this issue which is general and not specific to this one program? More importantly, is there a HOWTO or FAQ or whatever that details working with these libraries—lic5, libc6 (2.0 and 2.1), lisbstdc++ and **egcs**? Unless this information is readily available to newbies so that new software can be added to their Linux system, they will be dependent on the software provided by their distribution. I would appreciate your solution to this problem. —Don, dollberg@worldnet.att.net

It seems that you have the libraries you need, but that Caldera and the package you're trying to install (probably built on Red Hat) don't agree on dependency names. You can try to force the install with

```
rpm -i --nodeps package.rpm
```

Another option is to download package.src.rpm and rebuild it:

```
rpm --rebuild package.src.rpm
```

This will generate an rpm that will install on your system. —Marc Merlin, merlin@varesearch.com

Ejecting a CD-ROM

I have Red Hat 5.1, 5.2 and 6 and I can't get my CD-ROM to eject. I have tried umount, then eject on the CD-ROM. I've tried umount, then eject on the cdplayer window. I've also tried it mounted; nothing works. My system is a Compaq Deskpro 4000 pII CPU, 3.2GB hard drive, 32MB RAM, IDE CD-ROM. —Uriah Seagraves, useagraves@excite.com

When the CD-ROM is mounted, the operating system will issue a lock command that prevents the eject function from operating. I suspect your CD-ROM is not actually being unmounted. Execute the command

```
cat /proc/mounts
```

to view what the kernel thinks is actually mounted to be sure the drive isn't in the list. —Chad Robinson, chadr@brt.com

The **eject** command uses the **cdrom** argument as default, check on your /dev directory if you have a symbolic link from “cdrom” to the actual device that handles your CD-ROM drive. In my case, also on a Red Hat 6.0 system, it is:

```
$ ls -l cdrom
lrwxrwxrwx   1 root     root   3 Aug  9 13:47 cdrom -> hdd
```

If there is no such link, create it by typing

```
$ ln -s hdd cdrom
```

It should work then. Actually, you can give any name to the CD-ROM unit by creating links on the /dev directory such as

```
ln -s hdd compactdisc
```

and later on use **eject compactdisc**. Since you do not mention the specific brand of your CD-ROM unit and it seems you have tested it under several Linux versions, bear in mind that the physical CD-ROM unit must support the **eject** command. Also, type **man eject** on your system and read more about the options of the eject command. —Felipe E. Barousse, fbarousse@piensa.com

Boot Process Question

How do I change the order of items in the boot process? I am running Red Hat on a laptop and I want to load the PCMCIA services before the initialization of the network interfaces. Right now, the interfaces are initialized and eth0 fails because the PCMCIA services don't load until later in the boot process. —Jeff Blaha, jeffery.f.blaha@us.arthurandersen.com

If you look at the /etc/rc.d/rc2.d directory and type **ls -la**, you will see a list of files (possibly among others) starting with a capital S exactly in the order they are executed by the boot process, and a list of files starting with capital K in the order they are executed by the shutdown or system going down processes. The order is alphanumeric and determined by the numbers that follow the Ss or the Ks. If you rename, say, S45pcmcia to S07pcmcia (i.e., to some number lower than the one belonging to the network startup), the PCMCIA services will start before the network services. This applies to all startup and shutdown script files on the rc1.d, rc2.d and rc3.d directories under /etc/rc.d/. Modifying startup and shutdown order of these files may damage your system behavior and/or some of its services, so be careful. —Felipe E. Barousse, fbarousse@piensa.com

This is how it's supposed to work: the Ethernet interface gets configured when pcmcia gets loaded, so there is nothing wrong with the message you see. However, should you need to start PCMCIA sooner, you can edit /etc/rc.d/init.d/pcmcia and lower the second digit on the chkconfig line. For instance, change **chkconfig: 2345 45 96** into **chkconfig: 2345 9 96**. Then type:

```
chkconfig  --del pcmcia;  chkconfig --add
pcmcia
```

The links in /etc/rc.d/rc?.d/ will be regenerated. This will change the order in which PCMCIA is started. —Marc Merlin, merlin@varesearch.com

Archive Index Issue Table of Contents

Advanced search

New Products

Ellen Dahl

Issue #68, December 1999

DupliDisk RAIDcase, REALTIME Product Suite, Aplio/PRO and more.

DupliDisk RAIDcase

Arco Computer Products, Inc. announced the DupliDisk RAIDcase, a real-time backup device offering PC users a simple and convenient way to maintain an exact, up-to-the-minute duplicate of their IDE hard drives. With the DupliDisk RAIDcase, any PC user can now have the security of a redundant drive that can take over instantly in the event of a disk crash. It requires no device drivers, is essentially OS-independent, and has been tested with systems running Linux among other platforms. Manufacturer's SRP is $435 US.

Contact: ARCO Computer Products, Inc., 2750 North 29th Ave., Hollywood, FL 33020, 954 925-2688, 954 925-2889 (fax), arco@arcoide.com, http://www.arcoide.com/.

REALTIME Product Suite

Advanced Management Solutions (AMS) announced its AMS REALTIME software on Linux, an enterprise project management suite that can support thousands of users without sacrificing performance. The AMS REALTIME products are ODBC-compliant and cross-platform compatible, providing a consistent look

and feel to all users and allowing seamless data interchange. Contact AMS for details pricing.

Contact: Advanced Management Solutions, 800-397-6829, info@amsusa.com, http://www.amsrealtime.com/.

Aplio/PRO

Aplio, Inc. announced its first IP Phone appliance for the SOHO and small business market. Aplio/PRO is a low-cost, easy-to-use stand-alone telephony device that routes telephone calls through the Internet, allowing users to make free calls anywhere in the world without a computer. It connects to the Internet using any existing Ethernet connection and plugs into a regular telephone, delivering state-of-the-art Internet telephony with superior sound quality. SRP is $299 US per unit.

Contact: Aplio, Inc., 1250 Bayhill Drive, Suite 201, San Bruno, CA 94066, 888-642-7546, 650-794-2759 (fax), info_usa@aplio.com, http://www.aplio.com/.

OpenLinux 2.3

Caldera Systems, Inc. shipped OpenLinux 2.3, the latest release of its OpenLinux distribution. OpenLinux 2.3 provides remote mass installation capabilities and has been Y2K tested. It is based on the 2.2.10 kernel and includes an improved LIZARD (LInux wiZARD) install with faster autoprobing. The package retails for $49.95 US and contains several updated features and commercial applications. Those who purchased OpenLinux 2.2 may upgrade to 2.3 for $19.95 US.

Contact: Caldera Systems, Inc., 240 West Center Street, Orem, UT 84057, 888-465-4689, 801-765-1313 (fax), info@calderasystems.com, http://www.calderasystems.com/.

Clustor 2.0

Active Tools unveiled a new version of Clustor 2.0 with a greatly expanded set of application programming interface commands for software developers. It implements a job distribution engine with API commands that allow one to incorporate Clustor easily within other applications. The Clustor 2.0 Bundle with one Clustor Root license and five Clustor Nodes licenses starts at $1995 US. Contact Active Tools for price quotes on custom configuration. A CD-ROM and/or printed manuals may be purchased for $20 US each. Free evaluation downloads are available from the web site.

Contact: Active Tools Inc., 246 First St., Suite 310, San Francisco, CA 94105, 415-882-7062, 415-680-2369 (fax), sales@activetools.com, http://www.activetools.com/.

V2.0 GNUPro Dev Kit for Linux

Cygnus Solutions announced an enhanced version of its GNUPro Toolkit for Linux application developers. GNUPro Dev Kit speeds up development time and generates greater software performance with pre-built binaries for easier installation and compiler optimizations for Pentium processor-based systems. New features include the ability to produce code optimized for Intel Pentium processors; extended C++ language support; an improved GUI installer; and a graphical debugger which now supports the debugging of multithreaded code. The GNUPro Dev Kit for Linux is priced at $79 US per copy.

Contact: Cygnus Solutions, 1325 Chesapeake Terrace, Sunnyvale, CA 94089, 408-542-9600, 408-542-9699 (fax), sales@cygnus.com, http://www.cygnus.com/linux/.

FortranPlus Explorer

N.A. Software Ltd. introduced their new FortranPlus Explorer pack. FortranPlus Version 2 provides full Fortran95 language, Fortran2000 extensions, OpenGL graphics compatibility and backwards compatibility extensions. The Explorer pack includes the full Version 2 compiler, GUI-based source level debugger and on-line documentation. Introductory single user prices for the Explorer pack, PC Linux version, are <\#163>95 pounds sterling/$150 US including VAT and shipping. Prices apply to PC (Intel processor or compatible) versions only. Multiple licenses are available.

Contact: N.A. Software Ltd., 62 Roscoe Street, Liverpool, Merseyside, L1 9DW, United Kingdom, +44-0-151-709-4738, +44-0-51-709-5645 (fax), marketing@nasoftware.co.uk, http://www.nasoftware.co.uk/.

ICEbox DS 2000, TA 2000 and NAS 1000

LAND-5 Corporation now provides an assortment of fault-tolerant disk RAID arrays and network-attached storage solutions for Linux resellers, developers and integrators. The “plug and play” ICEbox RAID and JBOD storage systems offer economical solutions for on-line mission-critical environments. The ICEbox DS 2000, a compact rack-mountable disk RAID storage system, performs parallel reads and writes on up to eighteen spindles and is scalable to several terabytes of on-line storage. For high-speed archiving, the ICEbox TA 2000, a high-performance tape RAID array can hold up to 1.6 terabytes of RAID 3 storage. The ICEbox NAS 1000 combines disk and tape technologies in the same chassis with up to 180GB of native storage or 144GB of RAID 5 disk storage with local tape backup. Disk RAID arrays can be purchased from $8,364 US, tape RAID arrays from $18,000 US, JBOD configurations start under $6,000 US, and network attached storage solutions are priced from $7,000 US.

Contact: LAND-5 Corporation, 9747 Business Park Avenue, San Diego, CA 92131, 888-226-6544 (toll-free), 858-566-3611 (fax), sales@land-5.com, http://www.land-5.com/.

LightningFAX v6.5.1

Interstar Technologies premiered its Fax Messaging solution, designed to allow one to send and receive faxes securely and accurately from a desktop. LightningFAX is a robust and reliable application for networks and is easy to install, configure, use and manage. It is a cross-platform solution which takes advantage of existing TCP/IP protocols, allowing leveraging of hardware and software across multiple platforms and throughout an organization. Version 6.5.1 includes the ability to run on a Linux platform. Contact Interstar for price quotes.

Contact: Interstar Technologies Inc., 5835 Verdun Avenue, Suite 302, Montréal, Quebec H4H 1M1, Canada, 514-766-1668, 514-766-1439 (fax), info@interstarinc.com, http://www.faxserver.com/.

OmniServer version 1.0

Omnicentrix Corporation released the OmniServer, a two-tier, multi-platform development and deployment environment combining the ease of BASIC programming with the rapid deployment capabilities of Java. The server software includes drag-and-drop form design, integrated database support and familiar language syntax assistants. OmniServer consists of a client and a server component; the server is a fully cross-platform solution and runs on Linux. It will run in any Java 1.1.6-enabled browser; one need not know Java to write

programs. Supported DBMS systems include Oracle, Sybase and SQL Server. A free demo is available at the web site; contact Omnicentric for pricing.

Contact: Omnicentrix Corporation, 198 Broadway, Suite 400, New York, NY 10038, 212-577-6664, sales@omnicentrix.com, http://www.omnicentrix.com/.

PCI-based S514 Card

Sangoma Technologies announced its new PCI-based S514 card, with significant performance improvements over the ISA-based S508, particularly in the area of bus throughput. The card is available in two versions: a dual-port version with one main 4Mbps port and one auxiliary 512kbps port (both ports fully support RS232 and V.35/X.21/EIA530), and a quad port version with two sets of interfaces as described above. North American retail prices including main-port cabling are $599 US for the S5141 with one CPU and a dual port card, $859 US for the S5142, with a dual CPU Quad port card. Drivers will initially be available for Linux, supporting frame relay and Cisco HDLC.

Contact: Sangoma Technologies Inc., 1001 Denison Street, Suite 101, Markham, Ontario L3R 2Z6, Canada, 905-474-1990, 905-474-9223 (fax), saleserv@sangoma.com, www.sangoma.com/news.htm.

PostShop, ScanShop, OCR Shop 4.5

Vividata, Inc. announced the release of version 4.5 of its PostShop, ScanShop and OCR Shop software products. New features include the addition of Heidelberg's ICC Color Management Module, support for new printers and scanners, easier network printer setup, compatibility with Red Hat 6.0, new libraries and new code. PostShop and ScanShop are available at $199 US for home use and starting at $495 US for corporate use. OCR Shop is available at $299 US for home use, starting at $1,495 US for corporate use.

Contact: Vividata, Inc., 1250 Addison St., Suite 213A, Berkeley, CA 94702, 510-841-6400, 510-841-9661 (fax), sales@vividata.com, http://www.vividata.com/.

Xess

Business Logic Corporation released Xess Spreadsheet for Linux Standard Edition Version 4.2, the first spreadsheet designed specifically for the X Window system and Motif. The Standard Edition of Xess is designed specifically for Linux users who want familiar spreadsheet functionality and ease of use on their Linux platform. For financial and science/engineering applications, Xess provides a full range of mathematical, statistical, financial, matrix and string functions. Xess Spreadsheet for Linux costs $69.95 US for a single-user license. Site licenses are available.

Contact: Business Logic Corporation, 3-304 Stone Rd. W., Suite 336, Guelph, Ontario N1G 4W4, Canada, 519-763-2097 ext.23, 519-763-5483 (fax), sales@blcorp.com, http://www.blcorp.com/.

Instant Extranet Server (IXS)

V-ONE Corporation announced its new Instant Extranet Server (IXS) which enables any size organization to benefit from secure business communication over the Internet. IXS automatically installs Linux v6.0 from Red Hat; set-up takes less than 30 minutes. IXS is simple to implement and use. It is seamlessly integrated with VPN security from V-ONE, providing a secure e-mail server, software for public and private web servers and FTP servers in one package. Regular IXS pricing is $1,495 US for a 15-seat license and $3,995 US for IXS Gold Edition, which features a 50-seat license. As a special introductory promotion, the standard 15-seat license is available for $995 US.

Contact: V-ONE Corporation, 20250 Century Boulevard, Suite 300, Germantown, MD 20874, 800-495-8663, 301-515-5280 (fax), sales@v-one.com, http://www.v-one.com/.

Archive Index Issue Table of Contents

Advanced search

Army National Guard Using Linux

Richard Ridgeway

Issue #68, December 1999

The Army migrates a war game tool from Hewlett Packard 700 series workstations using HP-UX to Intel-based Linux workstations.

The Army National Guard (ARNG) has long sought to port a military war game simulation to personal computers for cost reasons. After disappointing results porting the graphics portion of the war game to Windows 95/NT using Hummingbird's eXceed X Window Server, the developers were astounded by the performance using Linux. As a result, the ARNG ported the entire war game simulation to Linux. The Linux version is currently being fielded to more than 107 ARNG sites as well as to several foreign governments.

The SIMITAR (SIMulation In Training for Advanced Readiness) program was established by Congress as an Advanced Research Projects Agency effort in mid-1992 (Krug and Pickell, Army [ISSN 0004-2455], 46(2):57-59, February 1996). Part of this effort involved modifying a battlefield synchronization tool called Janus to train staffs and units at their hometown armories.

Cubic Applications Inc. (CAI), under contract by the Army National Guard, was tasked to port one of the sixteen programs making up the Janus war game from the HP-UX operating system to Linux. The selected program managed the visualization of the Janus war game on X Window System capable workstations. The primary reason for this tasking was to save money on the initial cost of the platform, and more importantly, to reduce continuing maintenance costs. CAI ported the graphic program written in C to three operating systems: Windows 95, Windows NT and Linux. The X-client environment in the Windows 95/NT operating systems was provided by Hummingbird Communications, Ltd, eXceed v5.11.

Table 1

Table 1 is a comparison of screen refresh times on different platforms and operating systems as one progressively zooms into a location having intersecting roads, barriers and mine fields within the Janus war game simulation. The graphic process was running locally on each platform. The center of each display was the same X,Y coordinate.

Note the large discrepancy between refresh rates at zoom level 6 among platforms and operating systems. The HP Apollo 715/100 workstation served as our benchmark to compare other systems against, because it was the fastest POSIX-compliant computer we had at the beginning of our testing. The range of times for the HP 715/100 was one to nine seconds.

The graphics performance of the P200Pro PC and P133 PC running Windows 95 was considered unacceptable for the live training environment of the ARNG. The refresh times at zoom factor 6 were 73 and 121 seconds, respectively. Discussion with the development support staff at Hummingbird Communications Ltd. revealed further increases in speed could not be achieved. The Windows 95 X server software had been optimized correctly, and the P200Pro PC platform was state of the art when these tests were performed. A Hummingbird staff member suggested the Windows NT version of eXceed might be faster than the Windows 95 version.

The graphics performance of the P133 PC running the Windows NT implementation of eXceed was also considered unacceptable. The refresh time at zoom factor 6 was 103 seconds. Compared to Windows 95, the Windows NT software performed slower at the extremes, but faster towards the middle zoom levels. The sum of refresh times for Windows NT was 527 seconds, compared to 511 seconds for Windows 95. Overall, the Windows NT port performed slower than the Windows 95 port. We were surprised to discover the Windows NT implementation did not function unless PC users were given administrator privileges within the Hummingbird software. It would not work at all if PC users were given ordinary user privileges. The privilege anomaly was previously unknown to Hummingbird. Granting super-user privileges to what should be normal users would be a significant security breach.

As a last resort, and actually only after a very casual mention by co-workers of an operating system found useful for Internet servers, we investigated the Official Red Hat Linux for Intel as another possible operating system for controlling the graphic display of the Janus simulation. Up to this point, we had never heard of Linux. To say we were suspicious of a free operating system is putting it mildly.

We were astonished at the graphics performance of the Linux operating system. The performance of an entry-level Pentium and a state-of-the-art

PentiumPro with Linux was faster than any previously tested platform, including the HP 715/100. The graphic refresh rates for a 200MHz Pentium Pro system ranged from less than one second to three seconds. The 200MHz Pentium Pro was three times faster than the HP 715/100 workstation (rated at 10.25 Xmark93 units). Even a 133MHz Pentium with a $90 graphics card was twice as fast as the HP 715/100, with values from less than one second to four seconds. As a comparison, the fastest HP workstation at the time of testing was rated at 41.13 Xmark93 units (C180-XP HP Visualize-EG using the PA-8000/180MHz processor, retailing at around $51,400).

Apparently, the combination of CPU horsepower, accelerated graphic cards and reduced operating system overhead were responsible for these dramatic results. In terms of X performance alone, several high-performance accelerated graphics cards for the Intel platform outperformed the fastest HP X station at the time of testing.

The mid-performance graphic card in the 200MHz Pentium Pro Linux workstation has been benchmarked at 112,000 Xstones (1280x1024, 135MHz dot clock rate, 8 bits per pixel, data from www.goof.com/xbench/summary.html). State-of-the-art graphics cards are capable of almost 300,000 Xstones. For example, the Diamond Stealth 64 with the S3-968 chip set (PCI bus, 135MHz dot rate, 1280x1024 resolution, 8 bits per pixel) was rated at 291,000 Xstones in a 133MHz Pentium Linux workstation with 16MB of main memory. As a comparison, the Hewlett Packard series of X stations have been rated by HP as follows: Entria Color at 106,000 Xstones; Entria Hi-Res Color at 135,000 Xstones; Envizex “a” series at 168,000 Xstones; and the fastest Envizex “p” series at 220,000 Xstones.

The decision to use Linux workstations was driven by cost and performance parameters. In terms of cost, the Intel-based Linux workstations were much cheaper than those from other POSIX-compliant workstation providers. A 133MHz Pentium-based Linux workstation (214 MIPS) with 32MB RAM, 2GB hard drive, accelerated graphics card, 20-inch color monitor and a LAN card cost around $3,000. The HP 712/100 (122MIPS) with 32MB RAM, 2GB hard drive, 20-inch color monitor and a LAN interface cost around $10,600 (SEWP contract). At that time, the entry-level HP 712/64 (78MIPS) with 32MB RAM, 1GB hard drive, 20-inch color monitor and a LAN interface cost around $12,000 (GSA price).

In terms of the performance/cost ratio (higher number being better, performance being compared to the HP 715/100), the P133MHz Linux workstation is 2.25/$3K = 0.75, with the 715/100 being 1/$10.6K = 0.09. The performance/cost ratio of the P133MHz Linux workstation is 8.3 times (0.75/0.09) greater than the 715/100. In other words, the Linux workstation

gives you 8.3 times the performance per dollar spent as compared to the HP 715/100.

The Intel-based Linux workstation is cheaper not only in terms of initial cost and performance/cost ratio, but also in terms of adding/repairing/replacing components. To add/repair/replace Hewlett Packard/Sun/Silicon Graphics workstation components, you must purchase supported components. For example, adding 64MB of EDO memory to a Linux workstation at the time of testing cost $500 (price from SW Technology at http://www.swt.com/). Adding 64MB of memory to a Hewlett Packard workstation cost $3,800 to 5,600 (depending on the model, data from SEWP contract, CLIN 4.2.1 Memory). Adding a 4X CD-ROM drive to a Linux workstation cost $91 (price from SW Technology). Adding a 4X CD-ROM drive to a Hewlett Packard workstation cost $328 to $500 (depending on the model, data from SEWP contract, CLIN 4.2.4 CD-ROM Drives). Adding a 2GB hard drive to a Linux workstation costs $285 (Maxtor MX72004A, E-IDE). Adding a 2GB drive to a Hewlett Packard workstation costs $675 to $2,425 (depending on the model, data from SEWP contract, CLIN 4.2.2 Mass Storage Mini Towers and Disk Drives). Note that the SEWP contract prices are much less than the standard 22% discount Hewlett Packard gives to the government on non-contract purchases.

The Linux workstation has yet another advantage over more traditional graphics workstation providers—it can be a multi-boot platform. The partitions can be any combination of MS-DOS, Windows 95/98/NT, OS/2 or Linux. Linux includes the Linux boot loader (LILO) to specify which partition (operating system) to load.

Since the Army chose Hewlett Packard as the platform to develop and run Janus, no surprises would be lurking around the corner if the ARNG were to select the Hewlett Packard as the major Janus display platform. However, the cost/performance analysis argument, as well as continuing maintenance costs, argued against purchasing an entire suite of Hewlett Packard workstations.

In the end, the ARNG decided to purchase around 1700 Intel-based dual-boot Linux/Windows 95 workstations at under $1,800 each. The ARNG is using the Windows 95 side for CD-ROM-based Distance Learning training tasks. The Linux side is being used for the Janus war game simulation. Thus, the ARNG was able to get two disparate uses out of the same platform, with each use requiring a different operating system.

As the result of the above work, the ARNG decided to port the entire Janus simulation (16 programs) to Linux. The mostly FORTRAN with some C code software was successfully ported in about six man-weeks from HP-UX to Linux. Almost all of the port problems were related to flaws in the source code being

masked by the forgiving HP-UX FORTRAN compiler. The hardware being fielded to both execute and display the Janus simulation is now entirely composed of Intel-based Linux workstations. The port was performed by CAI's programmer, Kevin Buehler.

As one might imagine, the cost savings and performance increase did not go unnoticed in government circles. As a result of demonstrating the ARNG's complete port of Janus to the Linux operating system, the next release of Janus (v7.1) to the active duty U.S. Army will execute on Intel-based Linux workstations. Other war game simulations are currently being ported to Linux as a result of the ARNG's experimental program, using Janus v6.30 as the base source code.

We considered Linux as the last resort to solving a problem. As the good word spreads, perhaps others will consider Linux as the first resort. Linux saved the ARNG around $8,000,000 in initial purchase costs alone (not including the more expensive continuing maintenance costs). What might it save you?

In 1975, **Richard Ridgeway** wrote his first statistical analysis program on a Hewlett Packard desktop calculator with 20 registers. Dr. Ridgeway is currently working within the Tactical Directorate of the National Simulation Center at Fort Leavenworth, KS. He is working to improve a unique simulation (Spectrum) designed to train staff in military operations other than war. His wife, Luann, represents the 35th District in the Missouri House of Representatives (home.earthlink.net/~ridgeway(http://home.earthlink.net/~ridgeway). In his spare time, he rides horses and sails a bit. He can be reached via e-mail at ridgewr1@leav-emh1.army.mil.

Archive Index Issue Table of Contents

Advanced search

Transparent Firewalling

Federico

Christian Pellegrin

Issue #68, December 1999

The authors describe how to split an existing network without affecting the configuration of the machines already present by using the proxy arp technique.

One of the most difficult problems when dealing with a firewall is that the network or subnetwork we want to protect usually has to be split into at least two subnetworks: one on the external side and one on the internal, protected side. This, apart from the planning stage, can result in the reconfiguration of all machines in the network to the new configuration. What is worse is that in case of a hardware fault of the firewall, you'll have to reconfigure all machines in your network so they will be able to see the outside until you repair the firewall machine. The configuration of the firewall can be even harder if you don't have access to the configuration of the machine that actually connects your network to the external world, very often a router or something leased from a telco (telephone company).

We are going to explain a smarter way of adding a firewall to your network without breaking it into subnetworks or reconfiguring any machine on the internal or external network, except from the firewall machine itself, by just fooling the other machines into thinking nothing changed. We will cover the aspects of the network configuration and packet routing, not real packet-filtering firewalling, since this has already been covered in depth in another *Linux Journal* issue (see Resources).

Requirements and Assumptions

In the practical examples, we will pretend we have a C class network, where our contact with the external network is the .1 machine in the network (let it be a router, a machine or whatever). We will assume the IP numbers .2 and .3 to be

free for our use. We will also need another IP greater than or equal to .4, so we must be sure these aren't used for other purposes. Of course, with a different network configuration, you'll have to adjust some calculations to make the method fit your needs. To make things even clearer in the examples below, let's fix our C class network to the 1.2.3.0 one.

As for the Linux box that will work as a firewall, you must actually have the kernel compiled with the usual networking options, including IP forwarding and IP firewalling and whatever you else may need, for example, IP accounting. Also, enable IP forwarding in the kernel; in newer kernels, this is done by enabling it directly in the /proc file-system. (For example, by using **echo 1>/proc/sys/net/ipv4/ip_forward**. Your initialization scripts should take care of this.) Another thing to note is that you should have a working **arp** program installed on your system. In fact, some distributions are shipped with an arp binary that is compiled without the new SIOCSARP kernel interface, so it doesn't work, and if you forget to check this one thing, it may cause you many unnecessary headaches.

The Linux box must have two Ethernet cards installed: eth0, connected to the internal network; and eth1, directly connected to the gateway machine for our network to the world.

Network Configuration

Now, the network interfaces on the firewall machine must be configured. The internal network interface will be configured in the same way as the other internal machines, as if it had all machines on the network. So in our class C example, the eth0 configuration would be the following:

```
inet addr :  1.2.3.4
network   :  1.2.3.0
broadcast :  1.2.3.255
netmask   :  255.255.255.0
```

Remember that in place of **1.2.3.4**, you can use any other unused address. The commands to do this are

```
ifconfig eth0 1.2.3.4 netmask 255.255.255.0\
   broadcast 1.2.3.255
route add -net 1.2.3.0
```

This configuration will also apply to all machines in the internal network behind the firewall; of course, the IP address will change on each.

The second network interface, eth1, will be configured as if it had a very small network; actually, four IPs is the least we can manage. This is where the firewall machine and the gateway will be placed.

```
inet addr :  1.2.3.2
network   :  1.2.3.0
broadcast :  1.2.3.3
netmask   :  255.255.255.252
```

This is obtained using the following commands:

```
ifconfig eth1 1.2.3.2 netmask 255.255.255.252\
   broadcast 1.2.3.3
route add -net 1.2.3.0
route add default gw 1.2.3.1
```

The routing table is now set, so that packets for the entire class C network will be redirected to the internal eth0 interface, while the packets for the small network will be routed to the external eth1 interface. Finally, the default gateway has to be set for all machines on the internal network, that is, 1.2.3.1. Please note that this routing scheme works because the most specific route is applied first.

There are two more questions to solve. First, how will the gateway know how to reach the internal machines? Remember, we haven't changed the gateway configuration, so it still thinks it has the C class network attached to it. Second, how will the machines be able to reach the gateway? After all, they still believe they have the entire C class network, including 1.2.3.1, on their network interface. Well, it will be easy; we just make believe all the machines are on a class C network. The trick is to hack the process of translating the IP address to the hardware (in our case, Ethernet) address, which is called the ARP (address resolution protocol). (If you're not familiar with this, please consult the NET-HOWTO and Proxy-ARP miniHOWTO.) This can easily be done by telling our firewall machine to answer all ARP requests for the gateway on the internal network and reply to all requests for any internal machines from the gateway. Practically, this is done in two stages. First, by publishing, via arp, the gateway and the firewall machine on the internal network, more exactly:

```
arp -v -i eth0 -Ds 1.2.3.1 eth0 pub
arp -v -i eth0 -Ds 1.2.3.2 eth0 pub
```

Thus, when someone asks for .1 or .2 on the internal network, the firewall will reply, giving its Ethernet hardware address. In the second stage, we will publish all internal network IPs from .4 to .255, on the firewall-gateway small network. For the entire C class network, it look like this:

```
arp -v -i eth1 -Ds 1.2.3.128 eth1 netmask\
   255.255.255.128 pub
arp -v -i eth1 -Ds 1.2.3.64 eth1 netmask\
   255.255.255.192
pub
arp -v -i eth1 -Ds 1.2.3.32 eth1 netmask\
   255.255.255.224 pub
arp -v -i eth1 -Ds 1.2.3.16 eth1 netmask\
   255.255.255.240
pub
arp -v -i eth1 -Ds 1.2.3.8 eth1 netmask\
   255.255.255.248
```

```
pub
arp -v -i eth1 -Ds 1.2.3.4 eth1 netmask\
   255.255.255.252 pub
```

This way, we have partitioned the address space and published all our IPs. When the gateway asks for the hardware address of an internal machine, the firewall will reply giving its address. Since we turned on the IP packet forwarding, once the firewall has a packet and replies to the ARP request, it will forward it to the destination machine according to the routing table.

This solution has proven to be very useful for us, especially when we had to enhance the existing network without causing “too much trouble”. The last step is, of course, tuning the firewall, IP accounting, transparent proxy or whatever you need on the Linux box—but that is another story.

Resources

Federico is studying computer science at the University of Udine. When not hacking or coding he enjoys reading sf, listening to music and playing guitar. He can be reached at drzeus@infis.univ.ts.it.

Christian Pellegrin is studying astrophysics at the University of Trieste and works part-time as a system administrator and teacher in a high school. When not playing with Linux and other fun software or hardware he enjoys discussing who is the best film director of all times with his girlfriend. E-mails are welcome at chri@infis.univ.ts.it.

Archive Index Issue Table of Contents

Advanced search

Customizing the XDM Login Screen

Brian Lane

Issue #68, December 1999

How would you like your screen to look on start up? Here's how to make it look your way.

“What's an XDM screen? Is this more cryptic Linux geek speak?” Well, yes, but I'm going to make it easy to understand, so you too can speak more like a Linux geek. If you are running the X Window System and have your system set up to boot straight into X and display the box asking for your login name and password, you are already running XDM. If you are running X using the **startx** command from a shell prompt, you aren't running XDM now—but you will soon be.

XDM has features other than the ones relating to the xlogin box. These other features are useful only if you are running X on multiple screens or machines. If you are interested, read the **xdm** man page. In this article, I will focus on basic cosmetic changes like the background image, programs to be displayed while waiting for a login, colors and fonts used in the login box and the size and position of the login box.

I will assume you have X set up and running correctly. If you don't have X working, please consult the documentation that came with your Linux distribution.

If you already have XDM up and running, you can skip ahead to the section on customizing XDM.

Setting Up XDM

Setting up XDM requires you to change the run level of your system. The run level controls which mode the system is running in when it is rebooted. It can run in single user mode, multiuser mode without networking, multiuser mode with networking and multiuser mode with XDM running. My system is Red Hat

5.1 and it uses run level 3 for normal multiuser operation and run level 5 for XDM operation (multiuser, plus starting X at boot time). Edit your /etc/inittab file as the root user to change the run level of the system. First, make sure the XDM run level exists in /etc/inittab. It should look something like this and is usually located near the end of the file:

```
# Run XDM in run level 5
x:5:respawn:/usr/bin/X11/xdm -nodaemon
```

This is the entry from Red Hat 5.1. Slackware, Debian, SuSE and other Linux distributions with X should be similar. The run level number is 5 in this case, but may be different in your distribution.

You can test the XDM run level by typing **init 5**. If the login box appears and everything looks okay, you can change the default initlevel for bootup or experiment with the XDM changes without rebooting your system. If you don't want XDM to start at boot time, skip ahead to the next section.

Make a backup copy of the /etc/inittab file before you change anything. Rename it to something like inittab.bak.1, then look for the initdefault line, which is usually near the start of the /etc/inittab file. Since you are not yet running XDM, yours probably looks something like this:

```
id:3:initdefault:
```

To make your system start XDM at boot time, you change the 3 in this line to equal the number in the XDM run level line. In my case, I changed the 3 to a 5. Reboot your system, and a gray screen with a box in the middle asking for a user name and password will appear. You can log in and make sure everything is running okay, but that isn't necessary to complete this tutorial.

Customizing XDM

Now that XDM is up and running, we can start making changes. We will be switching between a text-mode login and the XDM screen. To get to the text mode console, press **<H>ctrl<H>-<H>alt<H>-F1**; to get back to the XDM screen, press **<H>ctrl<H>-<H>alt<H>-F7**. With some distributions, you may have to use **<H>ctrl<H>-<H>alt<H>-F6** for the XDM screen.

Change to text mode and log in as root. Change directories to /usr/lib/X11/xdm and look at the files present in this directory. These files control the behavior of your system when XDM is started and a user logs in using XDM. The files we are concerned with are:

- Xsetup (or Xsetup_0), which sets up the XDM screen
- Xresources, which controls the behavior of the xlogin widget

Changing the Background Color

Let's start by changing the background color to something other than gray. You can use any program which can display an image or color on the background, which is sometimes called the root window. One program included with the X distribution is **xsetroot**. Edit the Xsetup file and comment out any programs that may already be setting the background image, like xbanner, xv or xsetroot. Add the following line:

```
/usr/X11R6/bin/xsetroot -solid steelblue
```

Color names like steelblue are defined in the /usr/lib/X11/rgb.txt file. This maps color names to the actual Red/Green/Blue color settings, making things more readable. If you use a color name that has spaces in it, you need to enclose them in quotes, e.g., **"navy blue"**.

Save the Xsetup file and switch back to the XDM display by using **<H>ctrl<H>-<H>alt<H>-F7** (or **F6**, depending on which virtual console the X server is using for its display). Then restart XDM by pressing **<H>ctrl<H>-<H>alt<H>-<H>backspace<H>**. Note: do not use the **<H>del<H>** key. It will reboot the whole system instead of just restarting XDM.

You should now have a nice, solid steel-blue background. You can experiment with different colors until you find one that you like.

Changing the Background Pattern

A bitmap can be used to tile (copied over and over to cover the whole display) a simple two-color image onto the background instead of a solid color. There should be a collection of bitmaps in /usr/include/X11/bitmaps. You can also create your own using the bitmap program included with X windows. Try changing the xsetroot line to this:

```
/usr/X11R6/bin/xsetroot -bitmap\
/usr/include/X11/bitmaps/xsnow
```

Restart XDM as before, and you should now have a nice winter scene. You can change the foreground and background color with the bitmap by adding the **-fg** and **-bg** options and specifying a color. Try changing it to this:

```
/usr/X11R6/bin/xsetroot -bitmap\
/usr/include/X11/bitmaps/xsnow -fg blue -bg yellow
```

Not the most wonderful colors for snow, but you get the idea. The colors recognized by the **-fg** and **-bg** options are the same as the ones in the rgb.txt file discussed above.

You can also tile color bitmaps stored in the xpm format. The **xpmroot** program is used for this. Change the xsetroot line to something like this:

```
/usr/X11R6/bin/xpmroot\
/usr/include/X11/pixmaps/file.xpm
```

Displaying a Background Image

Now that we can display colors and tiled bitmaps on the background, it is time to display a picture on the background. To do this, I use a shareware graphics program called **xv**. You can get it from the xv home page at http://www.trilon.com/xv/, or it may be included with your Linux distribution. Remember, this is shareware, and you should support the author by sending him $25 if you find his program useful.

I have chosen to use xv, but any program capable of displaying an image on the background can be used. For xv, you tell it to display the image centered on the background. You also want it to exit immediately after displaying the image; otherwise, XDM will hang until the xv program is exited manually.

```
/usr/X11R6/bin/xv -root -rmode 5 -quit\
/root/.gromit01.jpg
```

I use this to put a picture in the center of the display. To view your changes, save the Xsetup file and press **<H>ctrl<H>_<H>alt<H>-F7** to switch back to the XDM screen. Press **<H>ctrl<H>-<H>alt<H>-<H>backspace<H>** to restart the X server. You should now see your image in the center of the screen, covered by the login box. **xv** supports several other placement options using the **-rmode** command. You can see a list of these options by typing **xv -rmode -1**.

Displaying Random Images

Listing 1

With the help of a simple Perl script, you can display a random image on the background each time XDM is run. Listing 1 is a simplified version of a script written by Scott Scriven, toykeeper@cheerful.com.

Type in this program or download it from ftp.linuxjournal.com/pub/lj/listings/issue68/3325.tgz. and save it as /usr/local/bin/bkgd. Make sure execute permissions are set by typing:

```
chmod ugo+x /usr/local/bin/bkgd
```

You may also have to change the path to xv and **find** to match your local setup. Including the absolute paths in the script ensures it will work correctly, even when the **$PATH** environment variable isn't set.

To load a random background, change the xv line in Xsetup to usr/local/bin/bkgd instead. Create a /usr/lib/X11/backgrounds directory and fill it with your favorite images. A couple of good places to look for background images are http://www.digitalblasphemy.com/ and http://ipix.yahoo.com/.

Listing 2 is a working Xsetup file with the intermediate steps commented out with "#" characters.

Listing 2

Customizing the Login Box

Now we want to customize the xlogin box using the /usr/lib/X11/xdm/Xresources file. This file is also used for configuring other XDM widgets like the chooser, but we aren't going to deal with these other options here—see the XDM man page to learn about them.

We can move the xlogin box, resize it, change its color, its fonts and what it says. I have moved mine into the lower-right corner and made it as small as I can, so that it doesn't cover up the background image.

The XDM xlogin widget uses X resources to specify these settings. They are all stored in the Xresources file and are read by XDM each time it restarts. A list of the available options, taken from the XDM man page, is shown in "Xresources Options". I will go through each option and explain its use.

I have ignored several more advanced xlogin resources, some of which may appear in the Xresources file. It is safe to leave them alone—the defaults set when you installed X should work fine. The XDM man page contains full descriptions of each option, if you want to experiment with them.

Move and Resize the xlogin Box

The **.width**, **.height**, **.x** and **.y** settings can be used to set the size and screen position of the xlogin box, or you can use **.geometry** to specify all of these at once. Let's move the login box to the lower-right corner and make it 300 by 250 pixels. The screen coordinates to be used start with 0,0 in the upper-left corner, and the coordinates of the lower-right corner will depend on your screen resolution. But X has another way to specify coordinates: **-0,-0** is the lower-right corner of the display, no matter what the screen size. Add this line to the Xresources file, near the other **xlogin*** definitions:

```
xlogin*geometry: 300x250-0-0
```

Save the file and restart the x server as you did after changing the background image. Your login box should now fit snugly into the lower-right corner of the display, revealing more of your background image.

Now we can change the colors. There are five resources relating to color in the above table. First, change the default foreground and background colors for the box using the **xlogin*foreground** and **.background** settings. Let's make it black on blue:

```
xlogin*foreground: black
xlogin*background: steelblue
```

Save and restart the X server to make sure your changes have taken effect. The greeting and login prompt did not change color, because you haven't changed them yet. You must specify each individual color you want to change. The **.greetColor** setting is the greeting that is displayed at the top of the box. **.promptColor** is the login: and password: prompt color, as well as the text you enter for your user name. **.failColor** is used for when the user name or password entered is invalid.

Try out these settings:

```
xlogin*foreground:  black
xlogin*background:  steelblue
xlogin*greetColor:  white
xlogin*promptColor: grey
xlogin*failColor:   red
```

Not a terribly inspiring color scheme, but better than black on white. Play around with it until you find the colors you like.

Changing the xlogin Fonts

The resources that control the four fonts we want to change are:

- **xlogin*font**: used for displaying the typed-in user name
- **xlogin*greetFont**: used to display the greeting
- **xlogin*promptFont**: used to display the prompts username: and password:
- **xlogin*failFont**: used for displaying that the login failed

Fonts under X are difficult to deal with. They have an abundance of options and modifiers, most of which are never used. The **xfontsel** program can make font selection much easier. Just browse through the fonts, selecting the font style, size and attributes you want. Then click on the select button and paste the font string into the Xresources file using your middle mouse button, or both mouse

buttons at once if you have a two-button mouse. Add these lines to your Xresources file:

```
xlogin*font:\
 -*-courier-bold-r-*-*-18-*-*-*-*-*-*-*
xlogin*greetFont:\
 -*-helvetica-bold-r-*-*-24-*-*-*-*-*-*-*
xlogin*promptFont:\
 -*-lucidatypewriter-bold-r-*-*-18-*-*-*-*-*-*-*
xlogin*failFont:\
 -*-times-bold-i-*-*-24-*-*-*-*-*-*-*
```

Experiment with the different fonts and sizes until you find something you like.

Changing the xlogin Prompts

You can also specify the text that is displayed for each of the four prompts associated with the xlogin widget. **.greeting** can be set to **CLIENTHOST** and will display the full host name of the system it is running on. The **.namePrompt** value is displayed to ask for the user name, **.passwdPrompt** asks for the password, and **.fail** is displayed when an unsuccessful login occurs. For example:

```
xlogin*greeting:         Welcome!
xlogin*namePrompt:       Name:\040
xlogin*passwdPrompt:     Password:
xlogin*fail:             !WRONG!
```

Add a Clock to Your XDM Screen

X distributions usually include the xclock program which can display a nifty looking analog clock. Add it to your XDM screen by inserting this line in your Xsetup file:

```
xclock -hl white
-hd white -bg black -fg white\ -geometry 100x100+0+0 &
```

This will display an analog clock of moderate size in the upper-left corner of the screen. The clock may stay running, even after a user has logged in.

Well, that's about it for the basic customization of XDM. There are many things to play with, and hopefully I have given you a good framework with which to begin experimenting. No two users have the same tastes, so it may take some time before you finally get the look and feel you want.

Xresources Options

Brian Lane and his wife Denise live in Olalla, Washington with their four computers. He spends his days developing embedded software and his nights writing Linux code. He can be contacted at nexus@tatoosh.com.

Kerberos

Cosimo Leipold

Issue #68, December 1999

Mr. Leipold explains what Kerberos is and why you want to use it.

Kerberos is a powerful set of programs which allow you to have encrypted connections to virtually anything: TELNET, FTP and even e-mail. This is of little use to the modem user, but in larger settings where Ethernet is used and sniffing is a real danger, Kerberos provides a viable and powerful solution. There is, however, one problem—Kerberos is notoriously known for being overly complex and difficult to install. This article is designed to help you make a good start; before you know it, with a little experimentation everything will (kind of) work. I wish I could explain everything in detail here, but then, this would be a book, not an article.

Here Comes Disaster

Like almost every major upgrade/change, it would be ideal to make a backup of your system. If you can, do it. If you don't have a tape drive be careful installing, Kereberos shouldn't write over your files, but if you want to be sure, make copies of /sbin somewhere. It takes two seconds and if something goes wrong it will be well worth your time.

Now we go grab the files. There are binary and source distributions of Kerberos for Linux. I've found that the source usually causes me less problems than the binaries. If you are up to playing around a bit go ahead and get the source, otherwise grab the binaries. Either can be found at http://web.mit.edu/kerberos/www/. I'll cover only the binaries here—simply because compiling the source isn't part of the scope of this article. I'm going to assume you managed to install things right. One important note: If you do choose to use the source (I recommend you do) make sure you extract all the tar archive files, not just the one containing the source. Also, if you install things in a different directory (not /krb5) then you will need to modify the files mentioned below to reflect your installation directory.

Two Files: /etc/krb5.conf and kdc.conf

These two files control virtually everything. They control the name of your REALM (see below) and who can connect; however, they can be fairly complex. Before you set them up, you will probably need to understand a few terms:

- **REALM**: I think of a realm as a "group". Machines will belong to this group. It has become almost standard procedure to make the realm the same as your domain name, just with capital letters. In my case, I called it UNDER, but you could call it anything you wanted. Yes, you can have more than one REALM, but you probably won't need one. Remember—the REALM *is* case sensitive! Pick a standard and stick to it.
- **KEYTAB**: a file that contains encrypted information allowing users/machines to authenticate themselves. Each machine that attempts to authenticate itself to the KDC (see below) must have one. This is done by issuing the **ktadd** command under **kadmin**.
- **KDC**: the Kerberos Distribution Center—the one that causes you headaches. This is the machine that controls access.
- **PRINCIPAL**: a principal is a "definition" of a user or a host. It is, effectively, what tells the server a user exists or a server is trusted.
- **INCIDENT**: when making a new principal, the notation is as follows: ***incident/host*@REALM**. For example, with ktadd, doing something like

```
ktadd host/pepsi.kellogg.nwu.edu@UNDER
```

- would make the incident host for the machine pepsi.kellogg.nwu.edu which is part of the UNDER realm.

Listing 1

Take a look at my /etc/krb5.conf file and notice the following:

- **default_realm = UNDER**: name of your realm.
- **profile = /krb5/var/krb5kdc/kdc.conf**: location of your kdc.conf file

Also note the section called **[realms]**. Under it, I have the name of my realm, UNDER, and the machine that hosts that information, in this case, underground.kellogg.nwu.edu. This will be the hostname of where you just installed Kerberos. **[domain_realm]** explains who can connect to the realm: anyone from anywhere within kellogg and within res-hall (the dorm rooms at Northwestern). Replace all the information mentioned above with the name of your REALM and the name of the machine you installed the server on.

Listing 2

Now on to your kdc.conf file. You are going to need to place it wherever you defined it in /etc/krb5.conf. My suggestion is you place it in the same directory I did. This will mean making these directories. If you do that, you can just copy my kdc.conf and save yourself some time. Just change the name of the realm to whatever you picked when making krb5.conf file. These two files are integral to making things work. You may want to double check for typos and possibly save yourself some headaches later.

Now, some tedious but easy work must be done. Create the database that controls who can login where by issuing the following command:

```
# kdb5_util create -r
Initializing database '/krb5/lvar/krb5kdc/principal' for
  realm '
master key name 'K/M@YOUR_REALM'
```

In my case, *YOUR_REALM* would have been UNDER. Just replace it with whatever the host name of the machine you are on right now is. You will be asked for a master password; pick something you won't forget.

Now you must make an ACL file. This basically controls who can connect and administer the REALM. It can also be complex, but for our purposes we will keep it simple. Edit the file (or create) defined by **acl =** in the kdc.conf file. Place the following on a line by itself:

```
*/admin@
```

This means the administrator can control things. I don't see why you would want anything else anyway.

More Fun with Commands

Now it is time to add users to the machine. First, start off with the administrator account.

```
# kadmin.local
kadmin.local: addprinc admin/admin@
Enter password for principal "admin/admin@
   your_password
Re-enter password for principal "admin/admin@
   your_password
Principal "admin/admin@YOUR_REALM" created.
```

Then create a keytab on the server. This will authenticate who can modify things on the server and who cannot. Make sure you place everything on one line (including **kadmin/changepw**):

```
kadmin.local: ktadd -k /etc/kadm5.keytab kadmin/admin kadmin/changepw
Entry for principal kadmin/admin with kvno 3, encryption type
DES-CBC-CRC added to keytab WRFILE:/etc/kadm5.keytab.
Entry for principal kadmin/changepw with kvno 3, encryption type
DES-CBC-CRC added to keytab WRFILE:/etc/kadm5.keytab.
```

You should see something *similar*, but probably not identical. Then you have to add the necessary information to the server. Edit your /etc/inetd.conf and insert the following:

```
krb5_prop 754/tcp # Kerberos v5 slave propagation
kerberos-adm 749/tcp # Kerberos v5 admin/chpwd
kerberos-adm 749/udp # Kerberos v5 admin/chpwd
kpasswd 761/tcp kpwd # Kerberos "passwd" -kfall
```

Now, as root, restart **inetd** and run **krb5kdc** and **kadmind**. Congratulations, most of the pain is over. It took me eight hours to get here my first time trying this—hope you did better.

Moment of Truth

Now test it. A few commands to know about are **kinit**, **klist** and **kdestory**. These initialize your tickets which authorize you, list them and destroy them. (Yes, from the user's point of view, everything is fairly simple.) So try it out by doing a **kinit admin/admin@YOUR_REALM**

```
underground:~> kinit admin/admin
Password for admin/admin@UNDER:
underground:~> klist
Ticket cache: /tmp/krb5cc_1000
Default principal: admin/admin@UNDER
Valid starting      Expires             Service principal
08 May 98 15:04:45  09 May 98 01:04:43  krbtgt/UNDER@UNDER
```

If you got it to work this far, you are virtually done. Add yourself as a user. Run kadmin—it should ask you for a password, same as the one you typed in way back when you created kadmin/admin. The procedure for adding another user is just as simple. Each user is a “principal” (don't ask me where the name came from).

```
kadmin:  addprinc
Enter password for principal "user@
Re-enter password for principal "user@
Principal "user@YOUR_REALM" created.
```

Should you make a mistake, just delete the principal like so:

```
kadmin:  delprinc user@
Are you sure you want to delete the principal "user@
Principal "user@YOUR_REALM" deleted.
```

Now test this out the same way you did the administrator. You should get a new ticket.

How Nice, But . . .

I still haven't explained how to *use* it, so here we go. In order for you to be able to use Kerberos encrypted services on a machine, it must satisfy the following:

- It has a principal host/*hostname*@REALM on the server

- It has the correct services set up.
- It has a keytab file and has /etc/inetd.conf set up right.

The easiest way to try this out is to set up the server so that it will let you make encrypted connections, before you attempt to add other machines. The problem is that it is a bit different from setting up another machine. So we are going to say we want to have kerberized TELNET and FTP on the machine pepsi.kellogg.nwu.edu for this example. To do this, you need to satisfy the three requirements above.

Let's go over the first. You are going to need to install Kerberos on the machine you want to offer kerberized services on first. All this means is putting the binaries on the machine (in our example pepsi.kellogg.nwu.edu). So just go install the binaries. You can, if you want, just copy them over. Then copy your /etc/krb5.conf file from the KDC (server) and place it on the machine you are giving kerberized services (pepsi.kellogg.nwu.edu). From that machine, you must run **kinit admin/admin**. Then run **kadmind** from your machine (or in my case pepsi.kellogg.nwu.edu) and run the following commands:

```
kadmin: addprinc host/pepsi.kellogg.nwu.edu
kadmin: addprinc telnet/pepsi.kellogg.nwu.edu
kadmin: addprinc ftp/pepsi.kellogg.nwu.edu
kadmin: ktadd host/pepsi.kellogg.nwu.edu telnet/pepsi.kellogg.nwu.edu
        ftp/pepsi.kellogg.nwu.edu
```

A quick explanation is in order. For each service you plan to offer that is kerberized, you must have a principal. Hence, the use of **telnet** and **ftp** with the **addprinc** command. Then you must make the keytab. That is done by issuing the ktadd command. All of this must be done on the machine, you are setting up to offer services (in this case pepsi.kellogg.nwu.edu).

Finally, edit your /etc/inetd.conf and add the following lines. You will want to comment out any previous definitions of telnet and ftp.

```
klogin stream tcp nowait root /krb5/sbin/klogind klogind -ki
   eklogin stream tcp nowait root /krb5/sbin/klogind klogind -eki
   kshell stream tcp nowait root /krb5/sbin/kshd kshd -ki
   telnet stream tcp nowait root /krb5/sbin/telnetd telnetd -a valid
   ftp stream tcp nowait root /krb5/sbin/ftpd -a
```

Go back to the main server and create yourself a ticket (**kinit *user*@YOUR_REALM**). Now make a user (see above addprinc command) for yourself and try to login using telnet like this:

```
underground:~> telnet -l cosimo pepsi
Trying 129.105.197.33...
Connected to pepsi.kellogg.nwu.edu (129.105.197.33).
Escape character is '^]'.
[ Kerberos V5 accepts you as "cosimo@UNDER" ]
Last login: Fri May  8 13:44:44 on tty2
Linux 2.0.30.
pepsi:~>
```

Note how you didn't have to enter a password. That's okay, because the ticket gave you the access to the machine. The ticket does eventually expire, but it can be renewed by issuing a new **kinit** command. (Do a **klist** to see when it does expire. If you copied my /etc/krb5.conf, it will be 600 minutes.)

Tie Some Loose Ends

Let us tie up a few loose ends: you should be aware that changing **telnetd -a valid** to **telnetd -a user** will allow users to login without authentication. If they don't run kinit, they won't even get a login prompt if you use **telnet -a valid**. Remember, since the passwords are stored on your KDC, make sure no one breaks into it; otherwise, they will have access to all the machines to which the KDC grants access. Get to know the terms principal, realm, kdc, etc.—almost anything you come across will use them.

Now What?

Well, you most likely feel I've left out a lot—and you are right, I have. There is plenty more to learn and plenty more to try. The MIT webpage has tons of links to more information. Of course, you can always e-mail me and ask me, and I'll try to answer you quickly.

Cosimo Leipold (cleipold@kellogg.nwu.edu) is a student at Northwestern University who has nothing better to do than learn UNIX. He now works for the Kellogg Graduate School of Management as a System Administrator. He lives with his love Chiara, who says he's a dork.

Archive Index Issue Table of Contents

Advanced search

What Can you Expect? —A Data Collection Project Using Linux

Denny Fox

Issue #68, December 1999

The author describes the end-to-end process of defining and implementing a data collection project using Linux. The project illustrates the use of Expect, stty, cron, a little C programming, gnuplot and ioctl to the serial device driver.

I have been doing a fair amount of testing and monitoring on the system and hardware clocks of my Debian 2.0 machine that I use for file (Samba), communication (ISDN/masq/diald), printing and modem pool (mserver) service on my home office network. I wanted to see how well I could correct the system clock with **adjtimex** without running **ntpd** all the time and keeping the ISDN line to my ISP up. I had been recording daily data for the clocks and an **ntp** (network time protocol) reference server using the logging feature of adjtimex with **cron** and an **Expect** script. I was noticing some odd changes from day to day, and was beginning to wonder if temperature was affecting the server's system clock.

Why Debian?

adjtimex allows you to fiddle with the kernel parameters which control the system clock. The file /etc/rc.boot/adjtimex contains settings for TICK and FREQ, the coarse and fine settings used to tune out variations in the frequency of the crystal oscillator on the motherboard which supplies the interrupts to the system clock timekeeping process. The command:

```
/usr/sbin/adjtimex --log --host ns.nts.umn.edu
```

logs data for the reference ntp server, in this case ns.nts.umn.edu, the system clock and the hardware clock to /var/log/clocks.log. By using

```
adjtimex --review=/var/log/clocks.log
```

you can get suggested changes for TICK and FREQ which will tune the kernel clock and hopefully get it to match the ntp reference server. This is all well and good, as long as the clock crystal is stable; but what if it varies with temperature?

The Idea

What if I could measure and record the temperature near or actually inside the server? I could then correlate the temperature data with the system clock data to see if they were related. I have a Micronta (Radio Shack) digital multimeter with a serial interface. All I needed to collect the data was a circuit to convert temperature to voltage and interface the meter to a serial port on the server.

Gathering the Hardware Tools

A little research on the Net turned up a couple of thermocouple to millivolt converters, but they cost much more than I wanted to pay. Being an electrical engineer and having worked at a measurement company for many years, I knew that a temperature to volts converter circuit is fairly simple. A couple of friends helped out by putting together a circuit that provides .01 volts out per degree Fahrenheit that fits on a piece of vector board about an inch square and runs from a 9-volt battery. See Figures 1 and 2 for the schematic diagram and a picture.

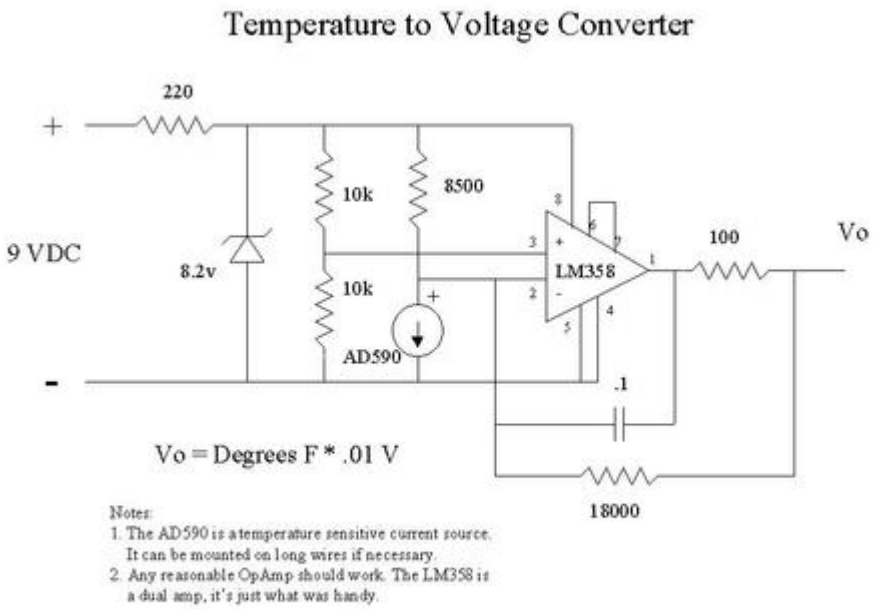


Figure 1. Schematic for Temperature-to-Volts Converter

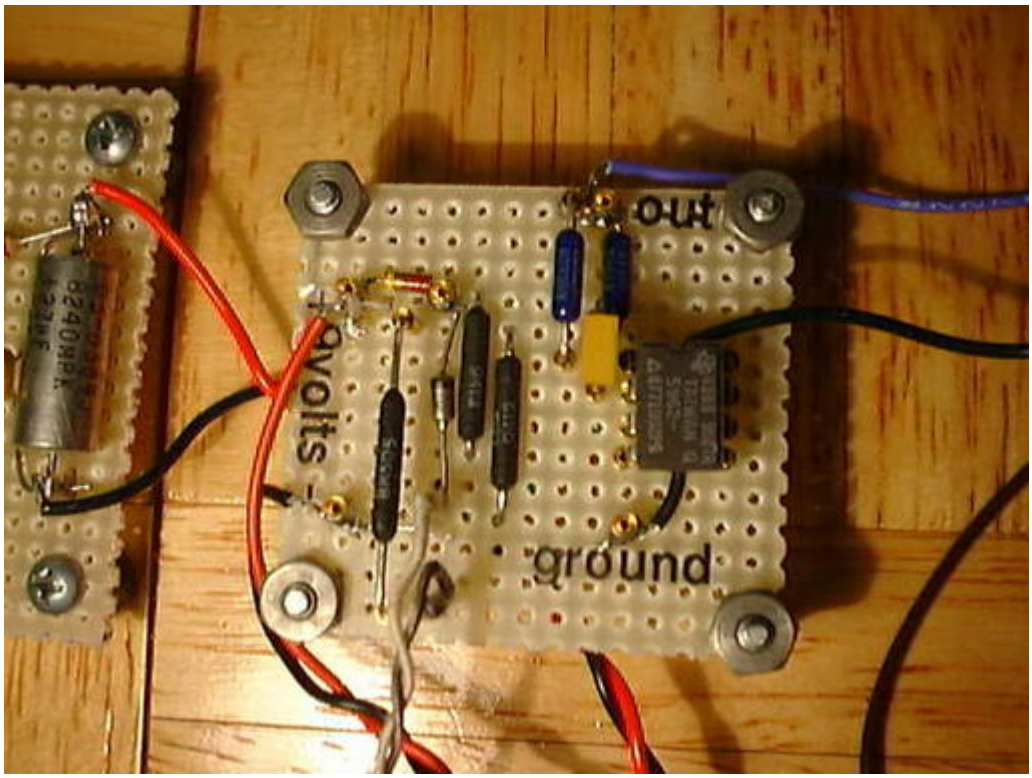


Figure 2. Converter

The Micronta No. 22-182 LCD Digital Multimeter comes with a five-conductor cable and a short section in the manual describing test programs for MS-DOS and MS-BASIC. A quick check showed that these worked fine. The serial interface communication parameters are 1200 baud, 7 data bits, no parity and 2 stop bits. Using the meter's continuity test function, I made a cable drawing as shown in Figure 3, a complete pinout and description of the serial port signals as found on a PC. These ports have male connectors with either 9 or 25 pins, and are wired as DTE (data terminal equipment). The RS-232 specification is designed so that a DTE port as on the PC can be connected to a piece of DCE (data communication equipment), typically a modem, with a straight-through cable. DCE usually has a female connector. You can use Table 1 to wire up a 9-pin to 25-pin conversion cable if you need one. Now I was ready to try reading the meter on Linux.

Figure 3

Table 1

Gathering the Software Tools

Before I hooked up the cable to /dev/ttyS0, I checked to see if there were any drivers like **getty** or **gpm** running on the port. Sure enough, there was a leftover gpm driver for a mouse, even though the mouse had long since been disconnected. I did an

```
/etc/init.d/gpm stop
```

and renamed the **init** script with

```
mv /etc/init.d/gpm /etc/init.d/nogpm
```

to prevent gpm from restarting in the event of a reboot. If you have a getty process running on the port, you will have to disable it by commenting out the correct line in /etc/inetd.conf and restarting init with

```
kill -HUP 1
```

Use something like

```
ps auxw|grep tty
```

to make sure the serial port you are trying to use is free.

I used stty to set the port to the meter's communication parameters with this command:

```
stty speed 1200 cs7 cstopb -echo clocal < \
   /dev/ttyS0
```

speed 1200 sets the baud rate, **cs7** sets 7 data bits, **cstopb** sets 2 stop bits and **parity** is **none** by default. I added the **-echo** to make sure the device driver didn't echo back characters sent to the port and **clocal** to disable the modem control signals.

I decided to use Expect to collect the data, since the meter has a very simple “send a command” and “get a response” paradigm. **Expect** is a powerful tool and can be used to automate UNIX programs which interact with a user or processes needing a command or trigger and return some kind of response. **Expect** is built on top of Tcl, a widely used extensible language. I had recently discovered Expect and found it is one of those tools you just don't know how you ever got along without. You can easily automate things with Expect that are extremely difficult or next to impossible with shell scripts or other languages. Sol Libes' book, *Exploring Expect*, was a valuable resource. Mr. Libes is the author of Expect. I also found the Expect and Tcl web pages very helpful. I had previously used Expect to automate a couple of tasks such as the clock data logging mentioned above.

The meter protocol is very simple: send the meter a **D\r** (a capital D followed by a return), and it sends back a 14-character string ending in a **\r** (return). The message sent back by the meter is of the form:

```
Byte     1 2 3 4 5 6 7 8 9 A B C D E
Ex. 1    D C - 1 . 9 9 9 9   V     \r
Ex. 2          1 . 9 9 9 9 M o h m \r
```

In practice, since this is a 3-1/2 digit multimeter, a space character replaces the least significant digit in column 9.

The First Try

Now that I had the port cleared and set to the right communication parameters and the cable hooked up, I was ready to talk to the meter.

However, when I hooked it up to the serial port on the Linux box, I got no outputs. Luckily, I have a serial breakoutbox, a piece of test equipment that has a two-color LED for each signal, switches to disconnect signals and sockets to jump them together. This is plugged in between the computer port and the piece of equipment you are trying to diagnose. My inexpensive box lights the LEDs red for negative voltages and green for positive voltages.

After much probing and watching the serial breakout box, I discovered that the meter depended on having the RTS (request to send) signal stay low to provide the negative voltage for the meter's output drive circuit. Without RTS low, the meter's TXD (transmitted data) line wouldn't work. Normally when you open a port, both the RTS and the DTR (data terminal ready) lines go high.

Creating the Missing Tools

Now, how do you control the modem control lines on a serial port? This is where having access to the source code for the serial drivers, and other utilities truly helped. If this were just a DOS application (single user, single tasking), it would be simple to read the ACE's (asynchronous communication element) control register, set the right bit, and write the data back out to the port. Since user-space programs can't write directly to system devices, I had to figure out how to tell the device driver to manipulate the RTS line. After much searching, I found a UNIX serial support site, which led to a serial utility site, which had a utility that I could hack to do what I wanted. I'm not a super C programmer, but this was just what I needed to give me the clues on how to operate the ioctl function of the serial driver. I hacked up a couple of programs: **clrrts.c** to clear the RTS line, and **modctl.c**, which can either set or clear RTS or DTR on a serial port. The source of clrrts.c and modctl.c can be found in the archive file ftp.linuxjournal.com/pub/lj/listings/issue68/3357.tgz.

Working Around the Gotchas

During my earlier sessions with Expect, I discovered a little hitch with Expect and cron. The Expect version 5.25 delivered with Debian 2.0 stable (libc6) will not spawn processes when run by cron. The Expect 5.19 on Debian 1.3 (libc5) works fine. I reported a bug to the Debian maintainers to learn that it might be a while until the libc6 issues were fixed. I worked around the problem by

manually installing the Expect 5.19 executable and Tcl 7.4 support libraries, from my Debian 1.3 system to the 2.0 server, which already had the general libc5 support libraries, to support another libc5 package I was running.

The meter has an auto-shutoff feature, which can't be disabled. It shuts the meter down if more than ten minutes go by with no activity. Clearly, this wasn't very good for long-term data collection. To fix this, I added some code to the Expect script to define how many times per hour I wanted the data logged, and set cron to read the meter once per minute. This keeps the meter on, but avoids having a huge log. The crontab line that runs rddmm.exp is:

```
* * * * * /usr/bin/expect5.19 /root/rddmm.exp
```

A couple of things showed up after a reboot. I discovered the Expect script was timing out, since the meter was not responding. Two things came out of this. The first was some interesting things that happen when you attempt to change certain **stty** parameters, and the serial port cable does not have connections to the modem control input lines: CTS (clear to send), DSR (data set ready) and DCD (data carrier detect). Basically, the port gets stuck. Since the cable that came with the meter left CTS, DSR and DCD open, and I did not want to modify the cable, I figured out which stty parameters *not* to use: *hupcl* and *crtscts*. I had placed *hupcl* in the original stty settings for the port while sorting out the RTS low requirement. The port had accepted the *hupcl* setting, because at the time the command was issued, I had the serial breakout box on the port and used the jumpers to wrap around the modem control signals. But when the meter cable alone was connected to the port, the lack of the feedback signals CTS, DSR and DCD caused *hupcl* to hang the port. This didn't show up until reboot.

Second, I needed to set “raw” mode on the serial port, the default as booted parameters are set to “cooked” which translates returns to newlines. This prevented the Expect script from seeing the \r at the end of the response. These changes were also incorporated into the Expect script.

The Expect script, **rddmm.exp**, with the lines numbered for reference, is included in the archive file along with a line-by-line explanation of the code.

Refinements

Both the temperature conversion circuit and the multimeter run from 9-volt batteries. Since I wanted to take data for weeks at a time without worrying about them going dead, I designed and built a couple of simple power supplies using adjustable voltage regulators and the cube transformers that plug in the wall, to act as battery eliminators. These also fit on a piece of vector board a little over one square inch. (See Figure 4.)

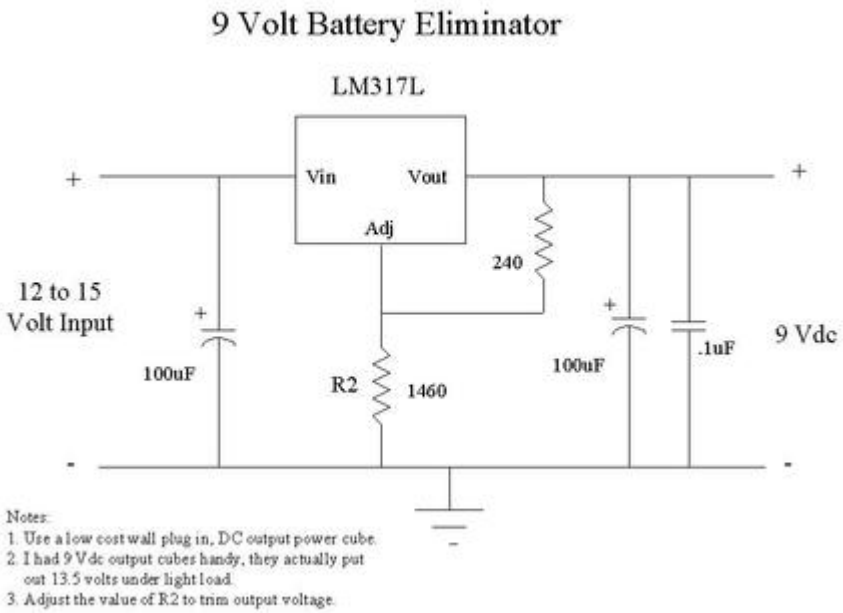


Figure 4. Battery Eliminator Schematic

Visualizing the Data

The last part was visualizing the data. I used **gnuplot** to read the log file and plot temperature versus time. I hadn't used gnuplot before, but a couple of hours going through the man pages got me to a point where I could view the plot on the Linux console or print it to my HP LaserJetIII.

Lines in /var/log/temps.log look like this:

```
Dec 31 10:45:01 server1 rddmm: 68.9 Degrees F
```

The operative gnuplot directives are:

```
set xdata time
set format x "%b %d\n%H:%M"
set title "Internal Server Temperature at Timekeeping Crystal"
set timefmt "%b %d %H:%M:%S"
set xrange [ "Jan 03 14:00:00" : "Jan 04 07:59:00"]
set ylabel "Degrees F" -2
plot "/var/log/temps.log" using 1:6 with lines 1
```

The **xdata** and **timefmt** directives tell gnuplot the horizontal axis is measured in time and how the times in the log file look. The **xrange** directive determines which lines of the log file get plotted. The **format x** directive defines the labels on the x axis; the **\n** between the date and time forces a two-line label. The plot command tells gnuplot where to find the log file, which columns to plot and to use line plot style 1. The **set title** and **set ylabel** put a title and y-axis label on the plot.

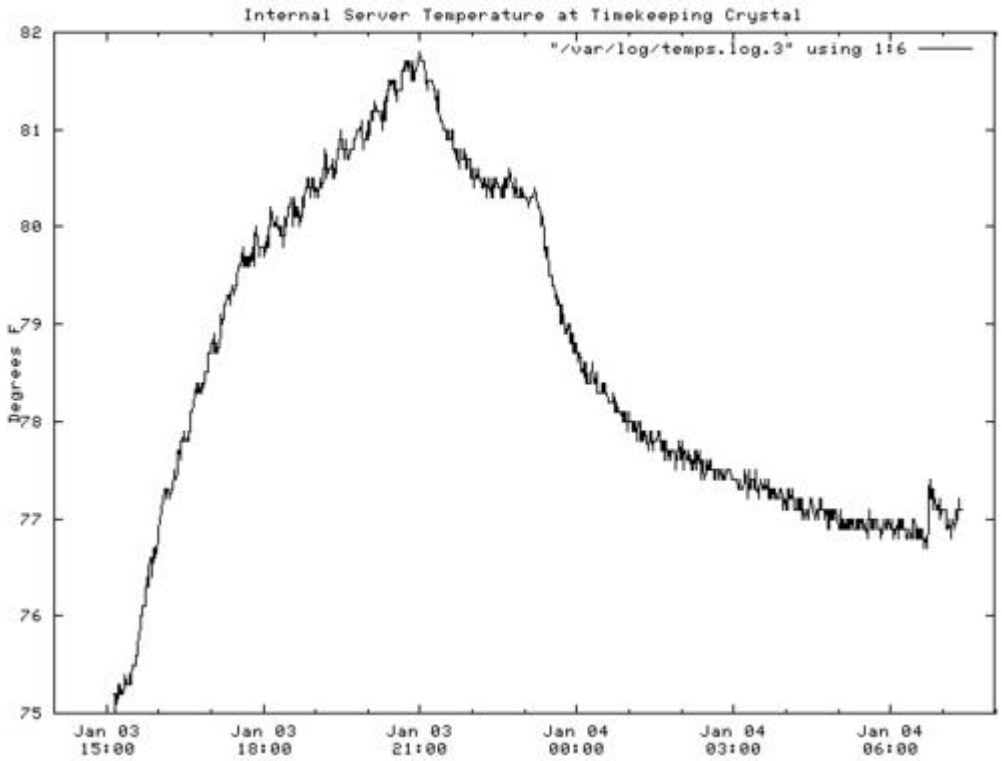


Figure 5. Temperature Versus Time

To print the plot to the laser jet, I used:

```
set terminal pcl5
set output '/root/plot.out'
replot
```

Then from the shell:

```
lpr /root/plot.out
```

In a similar fashion, I plotted the difference of system clock time measurements versus the reference NTP server on the Internet with the gnuplot directives shown here:

```
set xdata time
set timefmt "%Y-%m-%d %H:%M"
set xrange ["1999-01-03 15:00":"1999-01-04 07:00"]
set format x "%b %d\n%H:%M"
set title "Delta sysclock Minus Delta refclock"
set ylabel "Seconds" -2
plot "/root/clk_hr.prn" using 1:3 with lines 1
```

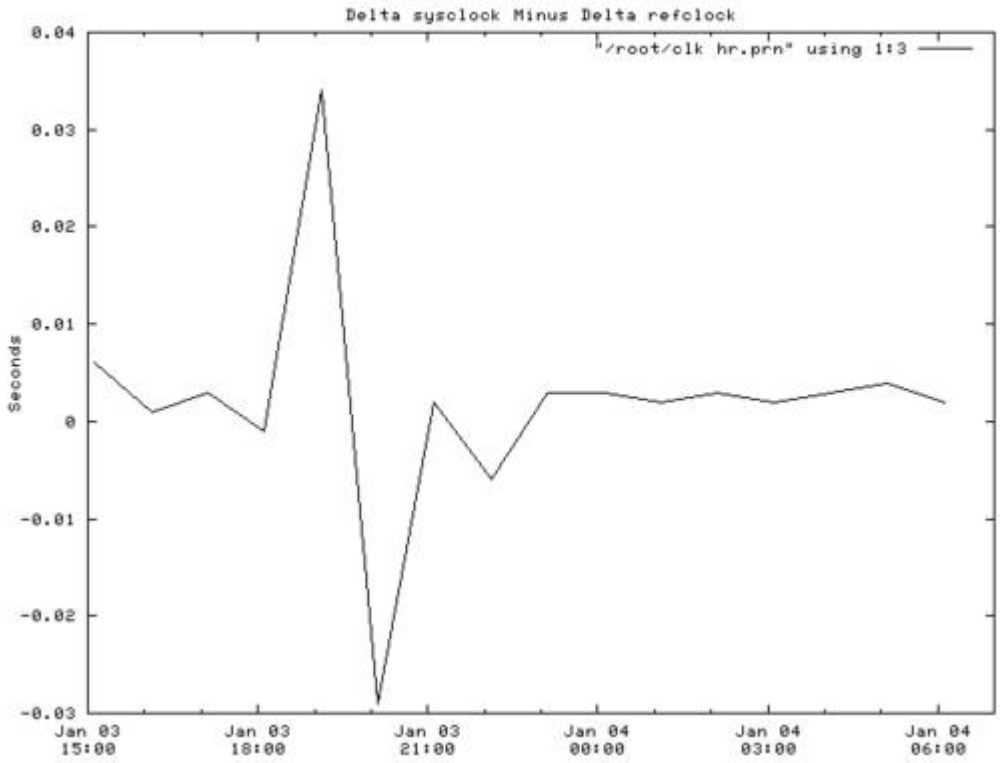


Figure 6. System Clock time Measurements Versus Reference Ntp Server

The Envelope, Please

And now, after all of this, is the system clock being affected by temperature? By looking at Figures 4 and 5, you can see that the variations in the clock differences do not follow the temperature variations inside the server. As a matter of fact, there is a large time variation that corrected itself, which I must chalk up to variations in network latency. By taking time data once per hour instead of once per day as I was originally doing, it became easier to identify the random network variations, which had originally peaked my curiosity.

Summary

These techniques are also adaptable to measuring and recording other physical parameters using devices with a serial interface. This particular digital multimeter can measure DC and AC voltage and current, capacitance, frequency and transistor gain. All these are accessible through the serial interface.

Linux and a project like this have brought me back to the days when you could actually create something useful with hardware and software. Sadly, most things for computers today come out of a shrink-wrapped box. Linux provides me with the tools I can use to make things happen in the real world.

Resources

Acknowledgements

Denny Fox has been active with designing hardware, software and auto-test equipment since the late '60s. When not hacking on something, Denny enjoys hiking, sailing, reading and playing guitar. The president of Micro Time Inc., he can be reached at dennyf@mninter.net.

Archive Index Issue Table of Contents

Advanced search

The Use of Linux in an Embedded System

Dave Pfaltzgraff

Issue #68, December 1999

One company's solution to a customer problem using Linux and open-source software.

I work for a company that's involved in the design and manufacturing of custom circuits. Historically, we have designed printed circuit boards and boxes that are further integrated by our customers into their systems. Until recently we were rarely involved in what is commonly called systems design and integration. However, times are changing and as we broaden our customer base, we have adapted to meet these needs. Most of the time, we have been able to meet the immediate requirements using DOS and in some cases Windows. This year, we began using Windows 98 to be able to use some of the newer video capabilities, and in one case, Linux stepped in to quickly solve a network connectivity problem. This paper is a brief description of one project that provided the opportunity to get more involved with Linux and the capabilities of this new operating system.

Description of the Problem

Recently, a customer, who is involved in retail sales and has customarily done their own systems integration, asked us to look at designing a Card Access System to replace their existing system. There were two primary reasons for this. One of these was cost. They felt that by designing a system to specifically meet their requirements, the production cost would be lower than by asking for customization of a product already on the market. The second reason was much more important to them. They wanted to be able to extend this new system and add interfaces into other parts of their overall store system. The store system includes their point-of-sale (POS) controller and a link to a security video system. These aspects of the system design will be discussed in more detail later.

The basic requirements to be met were:

- Entry into an area was to be controlled through the use of a credit-card-style *key*.
- If a user did not have their *key* it would be possible for them to enter the number manually.
- Employees could be added or deleted either automatically through the POS system or manually by the store manager.
- Access to an area was granted or denied based on the *key* and the type of area in question.
- Configuration of the system could be changed easily on site.
- If any area was accessed without a valid *key*, an alarm was to be sounded.
- Alarms could be reset by a manager, or optionally, by any valid *key*.
- Alarm conditions would be sent to the security video system.
- Managers would be able to override the system and allow doors to be propped open for extended periods of time.
- A must was the capability to query the system and examine activity based on area, time and/or employee.

Although these requirements are fairly basic, three things stood out. The first was they wanted the updates of the access database to occur from the POS system. However, this was something that remained “in the future”. Because of this, the second point was updates would need to be made locally which required an intuitive interface. Finally, the design of the security video system was being designed at the same time so the link to that would also be in the future. Other than having so many points that would remain for in the future, this was quite typical of many of the projects we've done. One other difference was we were to deliver a complete system rather than just the parts.

Implementation

Because of the distributed nature of the store environment, the decision was made to use an EIA RS-422 type of interface to distribute the system throughout the store. The basic idea was to have several serial lines from the central PC going out to multiple door-access modules on each serial line. Each door-access module would control up to four readers (or doors), see Figure 1. The choice of four readers was based on a typical proximity of doors and the amount of wiring needed for the readers, the solenoids and alarms. Since we had designed card readers and other data entry type of equipment in the past, it was decided to implement the card reader as a wall-mounted panel with an 8051 type device controlling it. Any entry from either the keypad or from a card swipe would be buffered and sent to the door-access module when polled. The door-access module would later forward it on to the central PC when it was

polled. The design of the system allows up to ten door-access modules per serial link or a maximum of 40 doors per serial link.

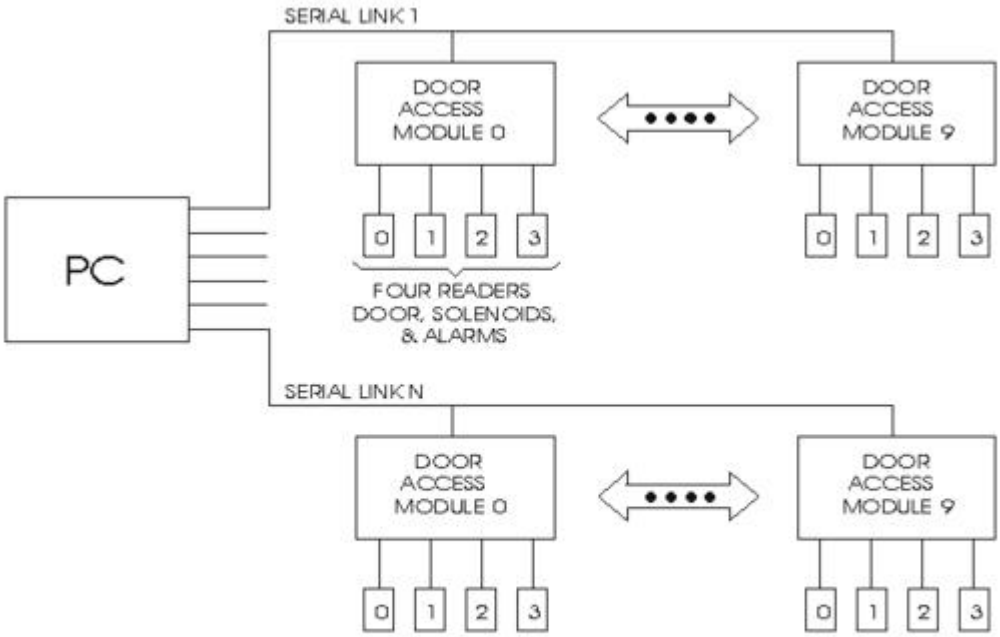


FIGURE 1: Hardware Architecture

Figure 1.

When the project was started, the feeling was that we would use Windows 95/98 as the operating system. This was a natural considering that in addition to the necessary communication it could provide database services through the use of Microsoft Access and an intuitive user interface. However, as the hardware design progressed, it became apparent the programming resources would not be available within an acceptable time frame. When it came time to allocate resources for this job, I was winding down from another job and was asked about how I would approach the design of the user interface. Not being very familiar with Windows programming but having some exposure to the browser interface, I asked if it would be appropriate to switch to Linux and provide the user interface through the use of Apache and an appropriate browser. The design team immediately picked up on the idea and presented it to the customer. When it was pointed out that this implementation would not require the user to be present at the equipment, which is typically crammed into a closet, the customer responded, “Neat.” When asked if they would have objections to implementing it on Linux, their response was, “Cool.” With this, I was on my way.

Committing to the Open Source Paradigm

Several aspects of using Linux needed to be considered before actually committing to this program. Many companies have been battling with these issues as Linux has struggled to move into the mainstream. The most important to us was whether or not the development resources would be available to us. Having worked with the GNU utilities, namely **gcc** and **gdb**, on a previous project, I felt developing the main program would present no problem. The next issue was the support required to allow the database interface to be provided through the user's browser. Again, a little research showed not only was the support there, but also there were many choices. One question that was asked of me was how we would handle the constant flux of kernel development. It didn't take long to realize it was not necessary to continually upgrade these systems just to stay up to date. All we needed was a workable version that could be propagated through the whole product. Since the source code is available, if the kernel were to develop in a direction incompatible with our system, we would not be abandoned with an albatross.

Implementing the Serial Interface and the Control Program

It seems that programming for a serial line in any system is a challenge and nothing comes easy. This was particularly true for this project as I had minimal experience with the UNIX environment and had to handle up to eight serial ports. Because of previous experience, the design team had already selected the Digi International eight-port 422 card. My first task was to set it up to run under Linux. I felt like I was groping around in the dark and looked for any kind of confirmation that I was doing things right. I tired to use **minicom** with a loop-back plug just to see if it worked. If it had, there'd be little mention of it here, but it didn't, and I spent considerable time reviewing what I had done. Finally, I called the Digi International support people. Of course, they had me try all the usual, and I was willing to entertain them on the off chance that I had overlooked something. To make a long story short, after several cycles of e-mail, one of their second-line support people picked up on my plight. While working with him, I certainly learned much about the inner workings of Linux, and it was worth the time for that alone. Finally, we discovered the problem was in the version of the driver I had; in fact, it had not been written to support the 422 card. A simple change and it started working. This experience proved to me that Linux is indeed a supported system.

Now that all the hardware was in place, it was my job to make it work. I was rather puzzled as to how I would handle up to eight serial interfaces and so I did some reading. This included the appropriate HOWTOs and any books I could get. Fortunately one of them was *Beginning Linux Programming* by Neil Matthew and Richard Stones, Wrox press, 1996. In it, they develop an excellent example of the use of FIFOs to communicate between tasks. I took their

example and expanded it, so that each serial interface had its own task and all communicated through FIFOs to the central controlling task. To see how the various tasks relate, see Figure 2. This proved to be an excellent choice for two reasons. The first is that the central controlling task could spawn a task for each serial link. If only one was needed for an application, only one task was spawned, etc. The second is that the serial task could perform all polling, error checking and retransmission without involving the controlling task in any way. This made the controlling task much simpler in that it dealt only with valid messages that needed action.

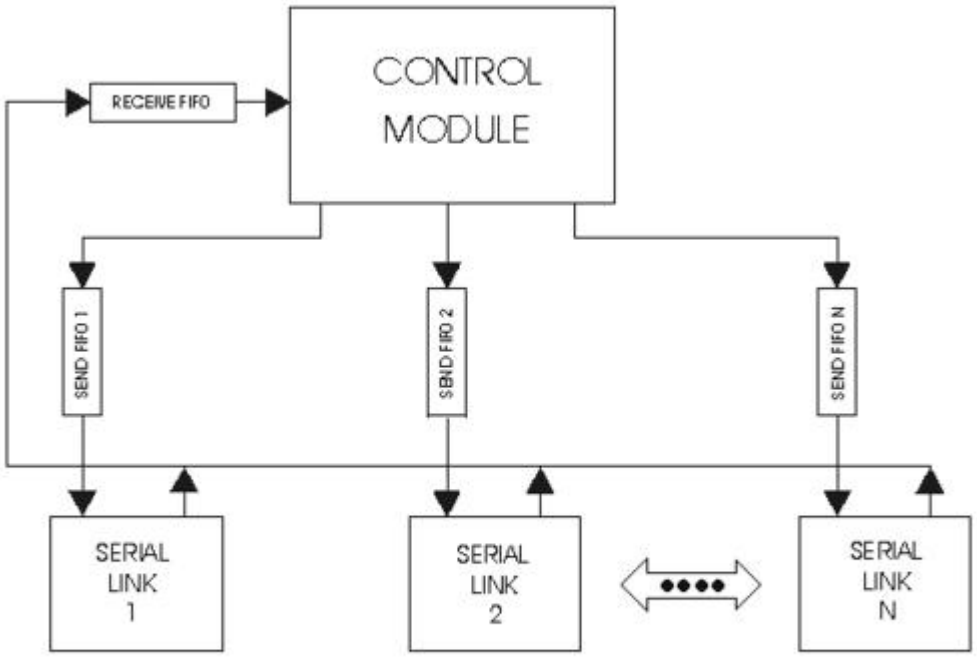


FIGURE 2: Structure of Serial I/O Modules

Figure 2.

One of the problems I've had in the past with programming serial links is that, unless you can master the interrupt mechanism, the system spends a large amount of time spinning its wheels waiting for something to happen. In the Linux system, I was able to use the **select** call to allow each task to go into the idle state until something needed to be done. This happens while the controlling task is waiting for a message to arrive from one of the readers. The timeout feature of the select call was also used to trigger a background task. If no messages arrived from any reader within ten seconds, a subtask would perform background housekeeping. It also happens in the serial task. In this case, the select is set to wake up on arrival of a message from either a reader or the controlling task. Again, the timeout feature was used, but this time it indicated that one of the door-access modules had failed to respond, which is indicative of a hardware problem.

With these basics in place, the development of the rest of the program progressed very rapidly. I want to point out that the multi-tasking power of Linux and the capabilities of gdb came through during this phase. Since the serial task was spawned by the controlling task, no screen was attached, and thus, there was no way to print the usual diagnostic messages during the development phase. One of the first things I did was to learn how to use **syslogd** to report error conditions within these tasks. Secondly, I used the capability of gdb to connect to an already running task and debug it.

Interface Between the Control Program and the Database

This was one area where we had just enough experience to be dangerous. We knew it could be done and this was one potential hot spot. In evaluating various database packages, we looked at the licensing terms and the availability of support for embedded and web applications. It seemed as though many of the databases available for Linux took care of the last two requirements. The licensing aspect was more varied in that several of the better-known packages provided for a single user or non-commercial use at no fee. Since we would be installing this on many systems and for commercial purposes, licensing was an issue we didn't want to overlook. When we pursued it, we found some of the license fees were higher than the going rate for MS Access. That alone eliminated some of the choices. We eventually settled on using PostgreSQL. Besides the very liberal copyright, we were strongly influenced by the availability of high-quality documentation. Having the opportunity to review the manuals before even committing to downloading the code was enough to reassure us this was a good path to follow.

Once the database was chosen, we had to get down to the details of implementation. The first part was to define the schema. Having had some experience with SQL previously, I found that **psql** was easily adapted. Knowing that the schema was likely to change as the project evolved, I wrote a few scripts to define all the tables and fill in default values. This took care of setting up the database. The next step was to define the interface to the control program. On the same premise that the database schema would change, I set up the interface as a separate module with subroutines to read selected data into working buffers. With the exception of the log table, all of the tables were read only for this function. Thus, there is a subroutine to read a selected record from each of the tables. The only other exception to this approach is that when the system first came up, a way was needed to determine just how many doors were implemented. This merely took a set of subroutines that would open the table, return the next entry in the table on each call and then close the table. This allowed the program to scan all possible entries to set up the communications links and other structures. All the functionality I needed was

provided by the lilbpq library included with the distribution. It turned out to be easier than I had first expected.

Interface Between the Database and the User Interface

This aspect of the project is a bit more complicated because of the number of different packages involved. If you recall from the outset, the idea was to use the web browser as the user interface. This meant we needed to provide a web server on our end of the link. For anyone in Linux, the choice of Apache is a no-brainer. However, we needed to link it up to our database in some way. Because of the availability of Perl expertise, we chose to go that route. In order to provide a complete package, we also needed to include other modules. For the interface to the database, we added ApacheDBI, DBD for Postgres and the Perl DBI package. To support all we needed in Apache, we added Digest, HTML-Parser, MIME-Base64, URI, Apache-SSI, libnet, libwww-perl, and mod-perl. Quite a conglomeration of packages, but when installed correctly, it all worked very well.

The design of the user interface was broken into two primary sections. The first was a set of routines that formats the data for display on the user's screen. The second is a main routine that accepts input from the user and processes it. Generally, that processing consists of updating the database and passing control to an appropriate routine to display a new set of data for the user. On top of all this, Javascript was used at the browser level to perform validity checking before passing parameters back to the main routine. Because of the amount of data being passed back and forth between the browser and the system, we kept things fairly simple. The only graphic we used was an identifying logo.

In general, the user is allowed to select a specific table from a top-level menu. Once a table is selected, the user is allowed to list the current table, look up an existing entry, modify an entry, and of course, add or delete entries. Since the data in the various tables may be somewhat sensitive and the people defining a door have different needs from those defining job codes, we added one other table to the system. The security table has nothing to do with normal operation of the system, but instead determines who may see what tables. It worked out nicely and it also gave us a means of allowing only certain users to have access to the system.

Conclusion

One of the primary questions to ask yourself at the end of the project is: Should we have done it differently? Although there is a lot of pressure to follow the Microsoft flow because people are familiar with that environment, I'd have to answer “No.” The system as we installed it has met or exceeded all of the

customer's expectations. The facilities provided with the linux operating system have allowed us to deliver a system that is reliable and is easily serviced and upgraded. For example, cron is used in conjunction with the logrotate utility to ensure that the log files don't fill up all disk space. Yet, at the same time, we can easily review the last four weeks of data. In a similar vein, PostgreSQL provides all of the database services that are required and, along with a quick script, delete stale data. These together keep the system from getting bogged down with bloated files. Other utilities come into play as well, the apcupsd daemon monitors the ups and ensures an orderly shutdown if necessary. As a side benefit, we also have a log of the power in the building. Using mgetty with pppd allows dial-in for any servicing that may be required. So far, all updates to the system have been handled through this path as the current network connection between us and the site is very slow.

The choice of using apache to provide the GUI couldn't have been better. When we started, the idea was to allow any browser on the network to gain access and allow updates to occur from remote locations. As noted above, the network connection is slow so this has not happened to any great extent. Also, by coincidence, the initial site did not have any other local machines. To take care of this situation, the X windows system was put into use and the Netscape browser is used at the machine. A special login was defined that placed the user into Netscape and on exiting X windows, the user is logged out. Security has been retained and the user is none the wiser.

Another thing to ask is: Where might we be headed in the future? The first step is provide a serial link to report activity to the external security system. A second is to have updates to the system database be done from another machine. Initial discussions centered around using a spare serial ports for these functions. No problem! In getting this far, we have all the experience with serial programming to know exactly what's needed for both tasks. Later discussions have moved towards using a TCP/IP link for the update function. Again, no problem with the networking environment provided by linux!

We have also been asked by the customer about setting up a central monitoring facility with the intent of being able report hardware failures. With the networking capabilities, both direct and dial-in, it was fairly simple to propose a system where a central unit could poll all of the units in the field. Alternatively, the field systems could call in and report their health. The system could even be a hybrid of these two approaches! The customer has not yet responded to this proposal, but it is not a big step to see that not only can all systems be monitored, but all program updates could be handled through this mechanism as well.

As you may see from my enthusiasm, I'm all for linux! The biggest barrier we've had is that of the familiar environment that people have come to expect. My expectations are that, given time, there will be a shift in the linux world and there will be 'education' of the masses and it will become more natural to use linux for projects of this type. In anticipation of this type of shift, the system was recently ported over to Red Hat version 6.0. With some minor tweaking for Apache and a recompilation of our code to use the new libraries, the transition was fairly uneventful.

Dave Pfaltzgraff is a Staff Engineer and has been involved in embedded systems design for over 20 years. He finds the openness of Linux to be a great pleasure and enjoys sharing “war” stories. He may be reached at dpfaltzg@patapsco.com.

Archive Index Issue Table of Contents

Advanced search

Building a Firewall with IP Chains

Pedro Paulo Bueno

Issue #68, December 1999

A quick introduction to the program ipchains.

Today, one of the most important topics in the computation world is security. How to improve security in a single or interconnected machine is sometimes hard to understand and difficult to implement. In this article, I will discuss how to implement a simple firewall on a Linux machine using IP chains.

History

IP chains could be new to users who upgraded their 2.0.36 kernels to the 2.2.*x* series, but old to those who worked in the 2.1.*x* series. **ipchains** is a rewrite of the well-known **ipfwadm**, which was a rewrite of BSD's **ipfw**, and was used to build firewalls in 2.0.*x* kernels. The are many reasons for this rewrite but perhaps the most important is ipfwadm couldn't allow protocols other than TCP, UDP or ICMP and it didn't handle fragments.

What It Is

Linux IP firewall chaining software is a program that uses the kernel IP packet filtering capability. A packet filter looks at the header of a packet and decides the fate of the entire packet. It can decide to **DENY** the packet (discard the packet as if it had never received it), **ACCEPT** (let the packet pass through), or **REJECT** (like deny, but notify the source of the packet).

Why You Want It

When you build your firewall you are looking for control and security of your network, and good firewall scripts are the key to this objective's success. If you are constantly receiving a **ping** flood from a specific IP address, you can deny all packets received from that IP, by creating a chain with this policy. **ipchains** is

able to read the policies of the scripts and give instructions to the IP packet filtering as to how to handle the incoming/outcoming packets.

Implementation

First, your kernel must be able to use IP chains. Look for the file /proc/net/ip_fwchains, if it exists, everything is okay. If not, you need to recompile your kernel and set these options:

```
CONFIG_FIREWALL=y
CONFIG_IP_FIREWALL=y
```

Next you need to know the syntax of ipchains necessary to create functional scripts. Let's imagine a hypothetical file called scriptf with some rules :

```
ipchains -N ippolicy
ipchains -I input -j ippolicy
ipchains -A ippolicy -p icmp -s 198.162.1.2 -j\
   DENY
ipchains -A ippolicy -p TCP -t 200.241.233.1
-j\
   DENY
```

This script will **DENY** every packet with the ICMP protocol from the specific source addresses (in our example: 192.168.1.2) and also **DENY** every packet with the TCP protocol where the target is the choosen address (again in our example: 200.241.233.1). Here's a step-by-step explanation:

- **ipchains -N ippolicy**: this line creates a new chain with the name **ippolicy**.
- **ipchains -I input -j ippolicy**: this line says all packets will be verified by **ippolicy** rules.
- **ipchains -A ippolicy -p icmp -s 198.162.1.2 -j DENY**: this line appends the rule **ippolicy** to the ICMP protocol packets, with a source address of 192.162.1.2 and denies them. Options are:
- -A: append one or more rules to the selected chain.
- -p: specify the protocol.
- -s: specify the source address (0/0 means all addresses).
- -j: specify the target of the rule, i.e., what to do if the packet matches it.
- **ipchains -A ippolicy -p TCP -t 200.241.233.1 -j DENY**: This line will append the rule **ippolicy** to the TCP protocol packets with a target address of 200.241.233.1 and denies them.

Now, you will want to inform the system that these rules are to be initialized at boot time. Remembering all information about system initialization is in the /etc/rc.d/init.d directory, copy the scriptf file to this directory and add a line like:

```
/etc/rc.d/init.d/scritpf
```

in the file /etc/rc.d/rc.sysinit to start it. An important option that could help you in the future is the **-F** flag, which is used when you want to create new rules and override all previous rules, that is:

```
ipchains -F ippolicy
```

Final Considerations

Ip chains is a very powerful tool that allows you to create many complex rules in order to protect your network. Just for fun, I built a small C program to build simple firewall scripts and simplify the rc.d process. It is open source and available at linuxgo.persogo.com.br/ipchains.html. Good places to get more information on how to build great scripts are the HOWTOs (see Resources). Read them before you start to build your own firewall scripts.

Resources

Pedro Paulo Ferreira Bueno, Science Computer Student from Catholic University of Goias (UCG- Brazil), is the manager of LinuxGO, the Goias Linux User Group and the network card moderator at Linux Knowledge base. He is a maniac linux user since he started with Linux in Kernel 2.0.7 . When he is not in front of his linux machine he is probability playing soccer. He can be reached at pedro.bueno@persogo.com.br.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Porting Progress Applications to Linux

Thomas Barringer

Issue #68, December 1999

An explanation of the work required to take an existing Progress application and deploy it on Linux, and the advantages and disadvantages of doing so.

Progress Software recently announced that by the time you read this article, the Progress RDBMS will be available in a native Linux port. Why is this good news? Because it immediately makes thousands of new applications available on our favorite operating system.

Over 5,000 applications have been written in Progress by Independent Software Vendors (ISVs). All of these applications could potentially use Linux as a back-end server, and many, if not most, provide a character interface which would allow Linux to serve as a client as well.

The Progress Environment

If you are not familiar with it, Progress is an enterprise-level relational database and 4GL development environment which has been running on UNIX systems for 15 years. It supports a wide variety of operating systems, including servers for NT, AIX, HP/UX, Solaris, Compaq Tru64 UNIX and many other popular platforms. It also supports character clients on UNIX plus character or graphical MS-Windows clients.

Version 8.3 of the database engine, the first version to be ported, can store half a terabyte of data in a database. (Of course, if that's not enough, you can connect to a couple of hundred databases simultaneously.) Also, each database can handle up to 4,000 simultaneous user connections. Version 9, which will be ported shortly after 8.3, supports 1000 times the data and up to 10,000 users.

One of the nice things about Progress, in my opinion, is that it also scales down. If you want to run single-user on a low-end 486, you can. Database size can go down to less than a megabyte. Running a single-user session, or serving a

handful of users on a small application remotely, can fit nicely in a 16MB Linux machine. Try running Oracle on NT and see what kind of power you need to get the same job done.

Deployment Architecture

Where does Linux fit into a Progress deployment? In order to answer that, you need to know something about Progress' architecture.

Databases can be accessed in either single-user or multi-user mode. Single-user sessions give a single client process complete control of a database. Multi-user Progress sessions consist of three different types of processes: clients, servers and a broker process which manages database connections. Deployment architectures can be chosen to support a wide variety of business needs and available machinery.

A broker process runs on the machine containing the database. The broker manages client connection requests and allocates a shared memory area which is used by all the processes connecting to the database.

The client process is the one the user sees; it presents information to and accepts input from the terminal. A client may also run as a background process, in which case it is not associated with a terminal but is architecturally identical to a foreground client. Clients are of two types: self-service and remote. A self-service client runs on the same machine as the broker and the database, and handles its own database manipulation requests. A remote client uses the TCP/IP networking layer rather than connecting directly to the shared-memory area created by the broker. This allows the remote client to be on a different machine than the database. The broker spawns server processes to respond to remote client requests. Each server process can handle many simultaneous client connections. The remote client sends data retrieval and update requests to the server; the server executes the requests as though it were itself a self-service client, and sends the results back to the remote client.

This design easily allows separation of database and clients. Not only that, the communications between them are standard so you can mix and match operating systems. If you are currently running Windows clients against an NT server, you can move the database to Linux and the Windows clients will operate unchanged.

New Deployment Options

This flexibility allows Linux to fill several different roles in a Progress deployment. Granted, the ability to replace the back-end server machine with Linux is a real boon in several ways, which any Linux fan will be able to tell you:

the smaller footprint, the difference in cost and the reliability of this operating system are already well-known. Also, remote maintenance comes free with Linux; you must otherwise buy not only the expensive OS license, but network-hogging third-party remote control software as well.

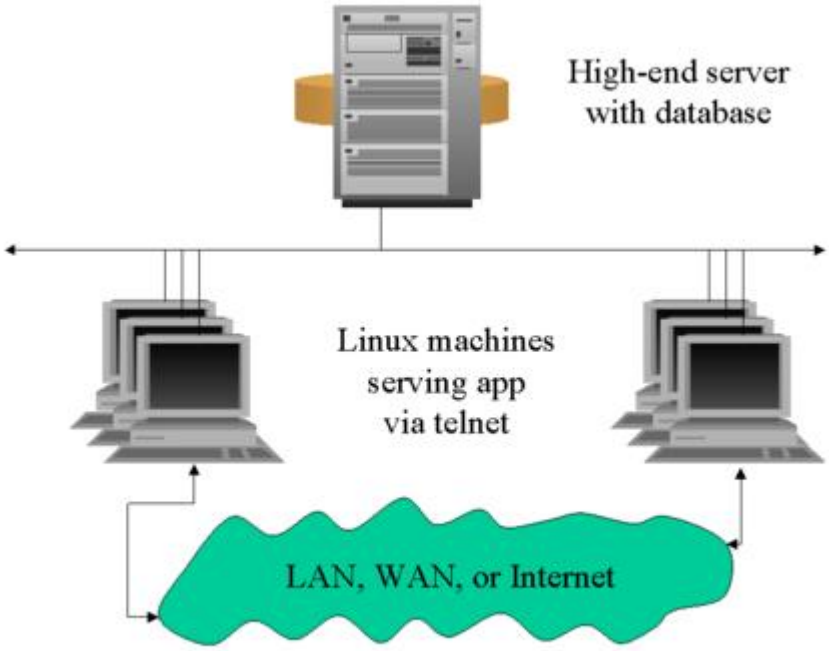


Figure 1. Linux can offload client work from an overburdened server.

Turn the situation around, and say you've already got a high-end UNIX server that 300 users **telnet** into in order to run their character-based application. If that single machine is struggling under the load of the database server plus running all the client processes, it is now easy to offload half the work, plus a fair proportion of the memory usage and disk I/O, by adding a few Linux servers to handle remote clients. Users no longer log in to your main UNIX machine; they log in to the Linux servers and use them as remote clients. Figure 1 is a diagram of this arrangement. Let's discuss four benefits of this arrangement.

One, the load on the database server is reduced. The memory consumed by running (say) ten clients on the machine is freed in exchange for one server process, which is about the same size as a single client. The number of clients a server process can support will vary based on the demands of the application. Server processes can typically serve the requests of anywhere from five to 20 remote clients simultaneously. Additional server processes are started automatically by the broker, based on the number of new connection requests from clients and on broker parameter settings. The CPU cycles which previously went to client calculations, screen manipulation and other front-end tasks are freed completely and moved to the new Linux machines. Any client-side disk I/

O, used for temporary storage, record sorting and application printing or other output, is also removed from the server. Using Linux can therefore boost the available power of your expensive high-end server machine.

Two, security levels are increased because the users no longer need to have a login on your main server. Instead, they now use TELNET to log in to one of the Linux machines that contain no critical data.

Three, if all the Linux machines are configured more or less identically, which would make sense from an administrative standpoint as well, you increase the fault tolerance of your architecture. If one of the Linux machines needs to come out of service for some reason, those who were previously using that machine for their clients can immediately log in to one of the other Linux machines and continue their work.

Four, the machines running Linux need fewer configurations than you might think. Given a generous four or five megabytes of memory per client and very little disk space, a dozen or more well-behaved interactive clients should be able to run on a single mid-range processor. This small footprint, along with the price of PCs today, means that these client services can be purchased for well under a hundred dollars a seat. Compare that price with adding a second high-end server to perform the same function.

The networking bandwidth required between the Linux machines and the database server will vary based on your application. You might choose either to put the Linux machines close to the database server or to distribute them to your field offices, depending on the application demands and the available infrastructure.

Linux may have a place at several levels of your deployment and can in fact be phased in as needed, or used only where it is convenient.

Porting Existing Applications

Since the front end (application) and the back end (the database) are so independent, let's examine what needs to be done one step at a time.

To move an existing Progress character-based application, the good news is all it takes is a recompile. That's it! In fact, there are some instances in which you don't even have to recompile; if you are staying within the same machine class, the same r-code (compiled code) can just be copied from one machine to another and it will run. This is because Progress r-code is not truly compiled code, but rather an interpreted binary format. As long as you remain within the same machine class (defined as machines with the same byte alignment, endianness, and word size), your compiled code will not even need a recompile.

The only time any complications in this process will occur is when you have OS-specific code within your application. Progress code doesn't care what OS it runs on, but if you make a call to an external routine (such as **ps** with flags specific to your old OS), you will need to make those modifications. However, this situation is fairly rare. Common operating system functions (such as file copy and delete) can be handled generically within Progress by using statements like **OS-COPY** which work regardless of the underlying platform.

In some circumstances, porting an existing database is just as easy. Officially, Progress does not support copying a database between operating systems or machine architectures. In order to do this, you must do a dump and load. This is a process of extracting the structure and all the data from the database in an ASCII form, then copying the ASCII data to the new machine and loading it into the new database. However, there is an unsupported loophole which I take advantage of regularly. Currently, I dual-boot my laptop with NT and Linux, and I have a Progress database on an NT FAT drive which is visible when I boot Linux. This database is accessible from either OS. I have application code (uncompiled because that is more convenient for me, but I could also share a compiled version) which accesses this database no matter which choice I make at boot time. There can't be much better proof than that. The true technical limitations on copying databases are much the same as those for copying compiled code: if the machine architecture and the word size are the same, you should be okay.

If your situation does require a dump and load, and you want the time your application is unavailable to be as short as possible, there are some techniques which will dramatically reduce the down time of your database. I regularly travel to customer sites and have seen some extreme examples of this: one database, with a size measurable in tens of gigabytes, was unavailable for only 45 minutes and only in the final phase of their dump and load.

This sort of performance, however, requires some custom work in advance. Generally, it involves dumping and loading stable data ahead of the actual cutover point; the greatest time savings occurs when you can define stable data most broadly. For example, if you have modification dates on a table, the entire table can be dumped and loaded in advance; then, when you are ready to cut over to the new system, re-dump only those records modified since the first dump.

A Good Match

Porting an existing character application written in Progress is remarkably simple. With over 5000 applications already written, the arrival of Progress on the Linux scene should make a large dent in the “Linux has no application” argument. Progress is flexible enough in its deployment capabilities that it

allows Linux to be integrated into existing deployments transparently to provide greater scalability and cost-effectiveness.

Tom Barringer is a Senior Consultant with Progress Software. He is most commonly found in airport lounges or in front of classrooms. What he does in his spare time is yet to be determined because he rarely has any. However, thanks to the wonders of modern technology, he can usually be reached at tomb@progress.com.

Archive Index Issue Table of Contents

Advanced search