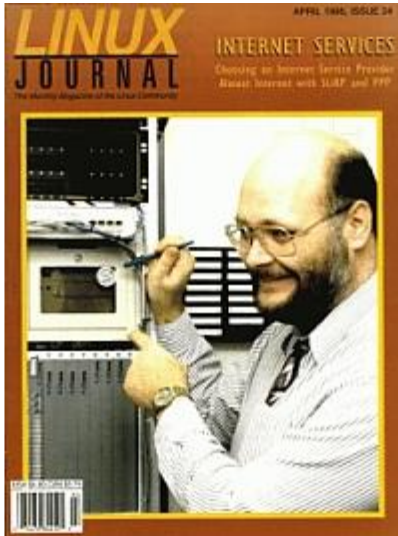
Linux Journal Issue #24/April 1996



Features

Choosing an Internet Service Provider *by Michael J Johnson*
: If you need to choose between a BBS, an on-line service, a shell account, and a PPP or SLIP account, read this informative article.

Almost Internet with SLiRP and PPP *by Jim Knoble*
: Jim guides the neophyte through installing and using SLiRP, a freely available software package which makes an ordinary shell account act like a SLIP or PPP account.

Building a Linux Firewall *by Chris Kostick*
: See how a Linux machine can be the guardian of your network's security.

News and Articles

XF-Mail
: by John M Fisk

Finding Linux Software
: by Erik Troan

The Trouble with Live Data
: by David Bonn

Columns

Letters to the Editor
Stop the Presses
Kernel Korner Dynamic Kernels - Discovery
Book Review Prime Time Freeware

New Products

Directories & References

Consultants Directory

Archive Index

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

Choosing an Internet Services Provider

Michael K. Johnson

Issue #24, April 1996

If you want or need some sort of connection to the Internet, but aren't sure what sort of connection, or where or how to connect, this article will answer some of your questions, and help you learn how to ask for answers to the rest.

Over a year ago, we ran a story about looking for an **I**nternet **S**ervice **P**rovider. Since then, the Internet has exploded in popularity, and the ISP scene has changed drastically. A PPP or SLIP connection is no longer considered a premium service for which you can expect to pay $100, or likely more, per month; now it is a basic service available in many areas for $20 per month or less. Many new ISPs have started up, and quite a few have gone out of business. More ISPs are available on a national scale. The "On-line Services" of the 80s are now connected to the Internet in one way or another. Your choices have expanded; so has your need for information about those choices.

If you want to know the difference between a BBS, an on-line service, a shell account, and a PPP or SLIP account, this article will tell you what you need to know (and perhaps more than you want to know). If you are worried about the relative dangers of connecting your computer to the outside world in different ways, a comparison follows. At the end, you will find some advice—a buyers' guide, if you will—that will help you choose between the ISPs in your area.

This article may initially put you into acronym overload. Acronyms are in such wide use in the Internet community that this is impossible to avoid. However, the acronyms are explained in the article, helping you speak the Internet language.

Capabilities and Limitations

For many years, people have operated **BBS**'s (Bulletin Board Systems) from their homes, and some BBSs have become so large that they have become businesses in their own right. Almost every BBS arranges for e-mail among its

own users, and many BBSs have been connected to a world-wide mail (and file) exchange network called FIDO. In FIDO, e-mail and files are passed from node (a single BBS) to node via telephone connections until they reach their final destination.

With this history of communication, it is not surprising that many BBSs chose to provide Internet e-mail as well. Some of them use a protocol developed for Unix, called **UUCP** (Unix to Unix CoPy), which works much like FIDO, and others use a direct IP (Internet Protocol) connection to the Internet, which can support telnet, ftp, and WWW connections as well as e-mail. Those BBSs that are connected with UUCP or direct IP usually provide Usenet news as well as Internet e-mail; those that have a direct IP connection usually support some Internet services as well.

Most BBSs have some sort of specialty, whether it's on-line games with other users, archives of some kind of programs, graphics, or discussions on one subject or another. More and more, they are also providing general Internet services like telnet, ftp, gopher, and WWW. Some allow you to call in using telnet from some other Internet site as well as through their own modems.

Most BBS systems provide you with menus which guide you through the provided services. The SysOp (System Operator) constructs the menus, determines which users have what privileges, and generally runs the show. Some large BBSs have multiple SysOps.

Most BBSs have very limited numbers of modems and phone lines, and therefore place significant limits on the amount of time a user can spend on line. Users are often granted extra time for doing things that are beneficial to the BBS, whether that's uploading a file, contributing to on-line discussion, donating money or equipment, or being the SysOp's friend. Some BBSs require you to pay for your time, but many operate on a donation or even free basis.

For years, the large-scale national and world-wide counterparts to BBSs have been the "**On-line Services**" such as Compuserve, Prodigy, America On-line, Genie, Bix, etc. While most of them originally started with menu structures much like those of a BBS, more and more have been requiring that the user run specialized software, most of which isn't available for Linux. You may be able to use DOSEMU to run DOS software, for example, but it is entirely possible that it will be easiest for you to reboot into DOS/Windows to connect.

Fortunately for Linux users, more and more of the on-line services have started to provide serial IP connections, which Linux handles natively, so specialized programs are not required. (See below for more on serial IP.)

Most of these services use large networks of modems all over the country or even world, which are often owned by some other company. Therefore, you may be charged several different fees for one connection: a fee for the time spent using the service and a fee for the same time spent using the modem, not to mention any long-distance fees that you need to pay to get to the modem.

Several of the on-line services have had a history of censorship of one form or another. Some even censor the private e-mail of their users. If this bothers you, ask about censorship before joining any service.

If you are comfortable with the command-line interface that Linux provides, you will probably be comfortable with a Shell account on a Linux or Unix computer managed by an ISP. Once your modem connects to the other modem and you log in, it is the same as an xterm session or a console login (without graphics capabilities) on your Linux box, except that the remote computer you are logged into is connected to the Internet. You usually use a standard Unix shell, with roughly the same choices available on your own Linux system, although some shell accounts also provide an optional menu interface similar to what a BBS provides.

In most cases, you are allowed to compile and run your own programs, if you wish. There is almost always a much larger selection of programs and services available than on a BBS. In addition, since you are connected to the Internet, all the services on the Internet are also available to you.

Unlike most BBSs and on-line services, you usually get to choose which program you wish to use to read mail and news. Pine, Elm, MH, mush, emacs-rmail, mailx, and other mail readers may all be available for you. You are likely to have your choice of rn, tin, trn, slrn, nn, and other news readers. Again, if you need to edit a file, you are likely to have your choice of editors. Emacs, vi, jed, and pico are common offerings.

If you want to browse the World-Wide Web, the text-based browser lynx will almost certainly be available.

A strong contrast to all the previous options, serial IP makes your machine a part of the Internet. For instance, instead of using ftp to transfer a file from a remote site on the Internet to your ISP's computer, and then using kermit or z-modem to transfer the file to your own computer, serial IP allows you to ftp the file straight from the remote site to your own computer, without storing the file somewhere as an intermediary step.

There are two kinds of serial IP in common use. The older, less advanced, less standard kind is called **SLIP**, which stands for **S**erial **L**ink **I**nternet **P**rotocol. The newer, standardized, easier to configure kind is called **PPP**, which stands for **P**oint to **P**oint **P**rotocol. PPP is designed to correct the flaws that were discovered in SLIP after it was designed. Linux supports both SLIP and PPP.

Besides the choice between SLIP and PPP, you may also have a choice between *static* and *dynamic* addressing. In order to understand this choice, you need to know something about the Internet. Each computer on the Internet is assigned its own unique number, which is called its **IP number** or **IP address**. No two computers on the Internet should be assigned the same address. When you configure your serial IP connection, one of the most important parameters is your IP address. So how do you get an IP address?

Each ISP has a large group of addresses that they parcel out one by one to their customers, and your ISP provides you with your address. There are two basic ways they go about doing this. One is to give you an IP address that is assigned just to you. In this case, called **static addressing**, you will always have the same address. The other way is for them to tell you what your IP address is when you connect. In this case, called **dynamic addressing**, that address is likely to change each time you log in.

In the same way, not only will your address (a set of four numbers like 10.24.105.27) change each time you connect with dynamic addressing, but your Internet hostname will change, too. Even if you have named your Linux system “ralph”, your Internet hostname might be dial12.city.isp.com one time and dial34.city.isp.com the next day. By contrast, with static addressing, you can nearly always get a reasonable Internet name such as ralph.isp.com.

From the standpoint of using common Internet tools like ftp, telnet, and WWW browsers, both static and dynamic addressing should act identically. However, from the standpoint of receiving e-mail, they have different potential. While you can get e-mail from both kinds of serial IP connections, only a static address and name allows you to create as many mailboxes as you like: me@ralph.isp.com, myfriend@ralph.isp.com, mywife@ralph.isp.com, myson@ralph.isp.com, and so on. By comparison, a dynamic address usually allows only one mailbox (usually something like *yourname*@isp.com), and you are not able to add new mailboxes on your own.

On the other hand, serial IP connections with dynamic addresses often cost less, and if you don't need the extra functionality, there is a certain amount of (almost) anonymity retained by not having the same Internet address each time you connect. Furthermore, setting up your own mailboxes (as is possible only with a static address) requires learning how to manage an **MTA** (mail transport

agent) such as **sendmail** or **smail**, which is not simple to do correctly. Even the *Linux Network Administrator's Guide* gives only simple introductions to configuring these two. By contrast, the “popmail” connections usually used for reading mail in the *yourname*@isp.com style of mailboxes are almost automatic to use if you are using a mail reader that supports popmail—and most do.

Either kind of serial IP account should let you browse Usenet news from home, using the **NNTP** (Network News Transport Protocol). However, this can be rather slow, depending on which news reader you use. The **xrn** (X-Windows based) and **slrn** (text-based) newsreaders are some of the few newsreaders which are still reasonably fast for reading news across a serial IP connection.

If you want users of other computers to be able to connect to your computer, serial IP is the way to do it. In particular, a static address and name will make this easy.

If you have a shell account, but want to use a graphical WWW browser such as Mosaic or Netscape, one option is to use a “**fake IP**” program such as SLiRP or TIA. These are programs that you run on your ISP's computer which talk to your computer using PPP or SLIP, convincing your computer that it is really part of the Internet. With these programs, you can ftp to a remote computer—but the remote computer talks to your ISP's computer, and the fake IP program acts as a “relay” for the connection.

This allows you to run multiple programs at the same time over the link, just like real serial IP. It allows you to be connected to the Internet. However, it does not allow others to connect from the Internet to your computer unless you explicitly allow it, which makes it a bit safer than a real serial IP connection if you don't pay close attention to security on your system.

It also has all the limitations of serial IP with dynamically assigned addresses and names, but in addition to those, it is not possible to run programs that use UDP, the User Datagram Protocol, across the link. The most common program that uses UDP is probably **talk**. In these cases, you normally telnet to another machine (probably the machine that you are running SLiRP or TIA on) and run the program from there.

Popmail and other services commonly available with serial IP services may be available over a fake IP connection, depending on your ISP.

Some ISPs prohibit the use of fake IP programs, on the grounds that you shouldn't need to do that if you have a real serial IP available, and that if you want IP, you ought to be paying for a real serial IP connection.

If a modem isn't fast enough for you, there are other kinds of connections. ISDN, Leased Line, and Frame Relay products are all available for Linux. However, these are expensive, and the capabilities are changing rapidly. Each would require an entire article to introduce its details.

Dangers

With a BBS, shell account, or on-line service, the dangers are small and minimal. The worst danger (possible with any service, of course) is probably a lightning strike that damages your modem, and possibly your computer. If you download programs directly from the service involved, you may need to think about viruses (none are known on Linux) or Trojan Horses, programs which claim to do one thing and really do another. If you consider censorship a danger, you are most likely to find it in various forms on the on-line services.

With fake IP, the dangers are also rather minimal, but the danger of a remote user breaking into your computer, while remote, now exists. Someone using the same ISP may be able to log into your computer if you don't pay attention to basic security, such as having good passwords for your accounts.

With serial IP, the dangers are very real. Unlike single-user program loaders like DOS and Windows, Linux is designed to function across the network. This power brings the potential for break-ins if security is not maintained. And in order to manage that power, it is best to know a little bit about what you are doing. The *Linux Network Administrator's Guide* (The NAG) tells you all you need to know to understand how to maintain your network, so have a copy handy.

Of course, you need to maintain good passwords—that should go without saying. Don't run unnecessary services. Read the /etc/inetd.conf file and comment out (by inserting a **#** character at the beginning of the line) services that you don't need. If you don't want anyone to be able to use telnet to connect to your machine, comment out the line that starts telnetd. If you are confused, read the NAG.

Make sure that you have a file called /etc/securetty. The securetty file controls which terminals root is allowed to log in on, and allowing root logins over the network is a good way to invite break-ins. Make sure that it does not contain any lines starting with ttyp--these are the *pseudo-ttys* that are used for network logins.

Pay attention to security alerts. Security holes in Linux or Linux distributions are announced on the comp.os.linux.announce newsgroup. Read these announcements and act on them. They contain careful, easy-to-follow instructions on how to close the security hole in question. If you have PGP

installed, use it to ascertain that the announcement is genuine before following the instructions.

If you have reason to be even more careful about security, take time to learn about authorization (**man tcpd**) and firewall software—there are several articles in this issue on that topic.

Competition

Pricing structures may have nothing to do with reality. If prices seem too low to believe, they may well be. How much trouble would it be to switch your e-mail to another site if your current provider goes out of business? What about your web site or your home page? Many ISPs have gone out of business in the last year, and the trend doesn't appear to be slowing down yet.

Do you want a local ISP or a national one? There are advantages to both; some people keep two accounts, one local and one national. If your ISP is near you, they will be easier to find. They may know more about people at the local phone company office and know who to talk to to get things done (or fixed) at a local level. By contrast, a national ISP has sites all around the nation that you can call in to, and by being bigger businesses, they may have more resources to provide the right distribution of equipment to meet all their users needs.

You need to decide what level of service you want, and be willing to pay for what you need. If your business depends on your connection, you probably don't want to go with a fly-by-night operation that may or may not be around next week, and may not have as much experience. If you just want to “surf the net” and can deal with trouble or switching providers if necessary, you are likely to be able to pay less—just don't expect the same level of service from all providers, regardless of price.

Also, it's important to understand that there are some things that the ISP can fix, and some things that the ISP can't fix any more than you can. The ISP can provide enough modems that you don't get a busy signal too often, and the ISP should be able to respond as quickly to emergency phone calls as they promise to, however quickly that is. The ISP can't, however, fix the rest of the Internet if their own connection to the Internet is broken by *their* provider, or make the phone company fix broken phone equipment on your schedule.

The Internet is constantly under construction. Choosing an ISP with multiple connections to the Internet through different providers makes it less likely that your ISP will temporarily loose Internet connectivity through no fault of their own. If your business depends on reliable connections to many different parts of the Internet, choose an ISP with two or three connections to major Internet providers, such as MCI, Sprint, PSI, or ANS.

Troubles will occur. Every ISP has troubles from time to time. You may have the choice of bouncing from one ISP to another. Our recommendation is to stick with one ISP as long as you can, especially if you choose a local ISP, because they can be extra helpful to people they know.

If you absolutely, positively need Internet access all the time, consider keeping a backup account at a different ISP. If you do this, make sure that the backup account is connected to the Internet in a different way than your main ISP. It may even be worth making a long-distance call in an emergency, depending on your need.

Speaking Their Language

The vocabulary covered in this article will help you speak your ISP's language. While many ISPs try to speak real English, knowing the occasional technical term can be very helpful, both in choosing an ISP and in working with your ISP once your choice is made.

You may know, or you may have learned in this article, more vocabulary than the help desk or sales staff know. If you sense that the person on the other end of the phone doesn't know what you are talking about, don't be afraid to ask to talk to technical staff.

See Internet Resources sidebar for more information.

Michael K. Johnson is the Editor of *Linux Journal* and full-time Internet user.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

Advanced search

Almost Internet with SLiRP and PPP

Jim Knoble

Issue #24, April 1996

Connecting to the Internet with Linux.

It's the '90s. More and more people around the world have heard of “getting connected to the Internet.” Some of those have actually made the technological, intellectual, and cultural leap into “Cyberspace”. And some of *those* are using Linux to get connected. I'm part of all three groups. In fact, I've become rather dependent on my Internet connection—it's like having a telephone, only more expensive.

To be connected to the Internet, you used to have only two choices: either a direct (and expensive) Internet connection, like those used by corporations and institutions, or a modem connection into somebody else's machine which had a direct Internet connection, using your computer as if it were a “dumb terminal”. You can still do that today, if you feel like it, but with the increasingly common availability of “real” net connections like SLIP (Serial Line Internet Protocol) and PPP (Point-to-Point Protocol), fewer Internet users really want to feel as if all they have on their desk is a dumb terminal. Yet, for a number of reasons, the only Internet access available to some of us is a dial-up dumb-terminal-ish Internet account, or “shell” account.

Enter SLiRP. SLiRP is a freely available software package, written by Danny Gasparovski (danjo@blitzen.canberra.edu.au), which makes an ordinary shell account act like a SLIP or PPP account. There are other so-called “SLIP emulators” available (one of the more notable of which is The Internet Adaptor —see References sidebar, as well as “terminal multiplexers” such as **term**, which uses a non-standard protocol between the shell account and the local computer. SLiRP has advantages over both groups, many of which, along with some disadvantages, the author details in the README file accompanying the SLiRP package.

The Remote Half: Installing SLiRP

SLiRP does not run on your “local” computer (the one that would otherwise be a dumb terminal), but rather on the “remote” computer (the one that's actually connected to the Internet). In order to install SLiRP properly, you need the following:

- A Unix shell account on the remote host
- A C compiler available on the remote host
- An editor that you know how to use on the remote host

If you're not sure whether you have a compiler or editor available, please contact your system administrator, or someone else who is familiar with your remote site.

The steps involved in installing SLiRP are as follows:

Getting SLiRP

The most recent version of SLiRP available at the time of this writing is slirp-0.95h.tar.gz. See References sidebar to find SLiRP. Once you have the package, “untar” it somewhere useful (such as /usr/local/src)--perhaps with the command **tar zxvf slirp-0.95h.tar.gz**.

Compiling the source code

Documentation of many sorts is included in the SLiRP package in slirp-0.95h/docs. Instructions for compiling SLiRP are in slirp-0.95h/docs/README.compiling, the gist of which is: change to the slirp-0.95h/src directory and run the configure program located there (usually by typing **./configure**); then build the program by typing **make**. If you have problems, consult the file slirp-0.95h/docs/README.getting-help.

Installing the program

When you have successfully compiled SLiRP, you will need to put the SLiRP binary somewhere where you can run it. If you don't already have a directory for your own programs, I suggest creating a directory called ~/bin, and then adding it to your **PATH** in your login scripts. If you're not sure how to do this on your system, check with your system administrator. Then, from the slirp-0.95h/src directory, perform the following commands:

```
strip slirp
cp slirp ~/bin
chmod 0700 ~/bin/slirp
```

(Alternatively, if your site has GNU `install' available, you may perform the above actions in one step with **install -s -m 0700 slirp ~/bin**.)

Configuring SLiRP

This is the tricky bit, partly because SLiRP is an evolving product, and its documentation is not necessarily entirely complete and up-to-date; however, I suggest reading slirp-0.95h/docs/CONFIG and slirp-0.95h/docs/README.ppp before doing anything else. Next, using your favorite text editor, create SLiRP's configuration file, called ~/.slirprc. At the very minimum, you will want to include a **baudrate** setting. If you're planning on using SLiRP to emulate PPP, you should also include a **ppp** flag, an **asyncmap** setting and **mtu** and **mru** values. Your .slirprc file might look something like mine:

```
baudrate 115200
ppp
asyncmap 0
mtu 552
mru 552
log start
```

You will most likely want to adjust the “baudrate” to suit your modem—some experimentation may be necessary. [“baudrate” should really be bits per second, or bpsrate. Even geeks who happen to know that their 14400 bps modems run at 2400 baud shouldn't set the baudrate parameter to 2400.. —ED]

Once you have SLiRP installed and configured, you can start it by simply typing **slirp**; SLiRP will then attempt to initiate a connection.

The Local Half: Installing PPP Under Linux

SLiRP is only half of your net connection; you also need to activate the other half of the connection on your local Linux machine. Linux, like SLiRP, “speaks” both SLIP and PPP—that is, support for TCP/IP networking over both SLIP connections and PPP connections is built into the Linux kernel. However, although both are available under Linux, I highly recommend using PPP instead of SLIP, for the following reasons:

- PPP is an Internet Standard Protocol—this means that it has undergone a standardization process approved by the Internet Architecture Board (IAB) and is an official part of the Internet Protocol Suite. SLIP, by contrast, is an “Internet non-standard” and is not on the standard track.
- PPP will work over some connections that are not 8-bit-transparent; SLIP will not.
- PPP can support authentication, peer address negotiation, packet header compression, and point-to-point error correction; SLIP can support none

of these (although Compressed SLIP, or CSLIP, does support packet header compression).

In short, it is my opinion that PPP is capable of providing a more robust and reliable connection than SLIP without significantly reducing the apparent speed of the connection.

PPP support under Linux comes in two halves. One half is part of the network system drivers, and comes built into the Linux kernel. The other half is a *daemon* process called **pppd**, which comes as a separate package.

Most distributions today come with PPP support built in, but if your Linux system is built from an older distribution, you may need to install PPP support. In order to properly install PPP networking under Linux, you need gcc installed, and either:

- A Linux kernel version 1.1.13 or greater with PPP support compiled in, or
- A Linux kernel version 1.0 or greater, plus either some experience compiling a Linux kernel or someone to help you do it.

If you don't know whether your kernel has PPP support compiled in, look for a message similar to the following when you boot up your machine:

```
PPP: version 0.2.7 (4 channels) [...]
```

You can also use the command **dmesg | grep PPP** to search the kernel boot messages for the above line. If no PPP messages exist, you will need to recompile your kernel to add PPP support (unless you are using the PPP kernel module; some distributions come with a PPP module ready to use). If you don't know what version your kernel is, use the **uname -a** command to display information about your system. If your kernel version is less than 1.1.13, you may need to recompile your kernel with a new PPP driver in order to use PPP. If you need to recompile your kernel, and especially if you need to install a newer PPP driver, see the documentation accompanying the pppd package and the Linux Documentation Project's **PPP-HOWTO** and **Kernel-HOWTO** for more information.

The steps involved in setting up PPP are as follows:

Getting pppd

The latest version of pppd at the writing of this article is ppp-2.1.2d.tar.gz. You may already have pppd; many Linux distributions include it. You can check for it with the following command: **find / -name "pppd*" -print**. If your distribution already has pppd installed, you can proceed to the configuration step. If you're

going to be compiling pppd, untar the source package somewhere useful (such as the place you untarred SLiRP).

Compiling and installing pppd

The pppd package comes with thorough instructions, including some specifically for Linux users, located in pppd-2.1.2d/README and pppd-2.1.2d/README.linux. I recommend you read both those documents and any documentation located in pppd-2.1.2d/linux before compiling and installing pppd. Then, change to the pppd-2.1.2d/pppd directory and create a Makefile using **cp Makefile.linux Makefile** (I recommend copying instead of linking so that you can make changes to the makefile if necessary without affecting the original). Inspect the makefile to make sure **BINDIR** and **MANDIR** are set to the correct location to install the pppd binary and manual page respectively (to comply with the Linux Filesystem Standard, they should probably be set to /usr/sbin and /usr/man). Make any required changes to the makefile using your favorite text editor. Build pppd by running **make**. To install pppd, first become the superuser (either by logging in as root or using the **su** command) and type **make install**; this installs both the pppd binary and the manual page. If you are replacing an older version of pppd, you may wish to make a backup copy of the older pppd until you are sure the new one works correctly. [If you are using the latest development kernel, you will need the latest version of pppd as well; it will be the latest in the 2.2.0 series, kept in the same places as the older 2.1.2 series pppd sources—ED]

Compiling and installing chat

The pppd package comes with a utility program called **chat** which performs “expect-send” scripts for dialing modems, performing automated logins, etc. chat is located in pppd-2.1.2d/chat/. The steps for compiling and installing chat are essentially the same as for pppd: copy the makefile, edit to suit your system, **make**, and then **make install** as the superuser. The chat manual page, however, needs to be installed by hand; use

```
install -m 0444 -o root -g man chat.8 /usr/man/man8/
```

to do this.

Configuring pppd

Now is a good time to read the manual page for pppd. While it is rather long, the manual page contains some information critical to understanding how to set up pppd, including explanations of several scripts called by pppd. There are

several steps involved in properly configuring pppd, and a few more in making it convenient to use:

- Creating the system-wide pppd configuration file /etc/ppp/options.
- Modifying the network configuration files to work with a PPP connection.
- Creating the scripts /etc/ppp/ip-up and /etc/ppp/ip-down to perform any needed actions when the PPP link becomes active and before it is closed, respectively.
- Configuring the system logging facility so that messages from pppd are written to the system logs.
- Creating scripts to start and stop pppd gracefully.

The system-wide configuration file, /etc/ppp/options, is the most important thing to get right for pppd to work. The file must exist and be readable by the root user, even if you don't plan to store configuration information in it—otherwise, pppd won't start. Unfortunately, there are so many possible options to configure for pppd that it's difficult to build a configuration file from scratch. I have put together an options file template for pppd, which includes each configurable option and explanatory text taken directly from the pppd manual page, and made it freely available for anonymous FTP (see References for locations). I recommend fetching the pppopt.tgz package to use as a starting point for configuration.

A configuration file for pppd looks like this:

```
## /etc/ppp/options -- config file for pppd
# async character map -- 32-bit hex; each bit
# is a character that needs to be escaped for
# pppd to receive it.  0x00000001 represents
# "<\>x00", and 0x80000000 represents "<\>x1f".
asyncmap 0
# Use hardware flow control (i.e. RTS/CTS) to
# control the flow of data on the serial port.
crtscts
# Add a default route to the system routing
# tables, using the peer as the gateway, when
# IPCP negotiation is successfully completed.
# This entry is removed when the PPP connection
# is broken.
defaultroute
# Set the MRU [Maximum Receive Unit] value to <n>
# for negotiation.  pppd will ask the peer to send
# packets of no more than <n> bytes. The minimum
# MRU value is 128. The default MRU value is 1500.
# A value of 296 is recommended for slow links
# (40 bytes for TCP/IP header + 256 bytes of data)
mru 552
# Disables the default behaviour when no local IP
# address is specified, which is to determine (if
# possible) the local IP address from the
# hostname. With this option, the peer will have
# to supply the local IP address during IPCP
# negotiation (unless it is specified explicitly on
# the command line or in an options file).
noipdefault
```

This file assumes that the modem is set to use hardware flow control (**crtscts**), that the link between the local machine and the remote one is 8-bit-clean (**asyncmap 0**), and that the remote machine will tell pppd what the local machine's IP address is (**noipdefault**). It also tells pppd to add a “default route” to the system routing tables (**defaultroute**), which is generally what we want for a dial-up Internet connection, and sets the “maximum receive unit”, the largest PPP packet that pppd will accept, to 552 (**mru 552**).

If you want to use software flow control instead of hardware flow control for your modem, you can use **xonxoff** instead of **crtscts**. You also need to add the XON and XOFF characters to the **asyncmap** option, as follows: the number following **asyncmap** is a 32-bit hexadecimal number, where each bit represents a character between 0x00 (**^@**) and 0x1f (**^_**) which must be “escaped”, or sent as a two-byte sequence to avoid getting swallowed or munged in transmission; **asyncmap 000a0000** escapes characters 0x13 (**^S**) and 0x11 (**^Q**), the XON/XOFF characters. You will also need to add this **asyncmap** setting to your SLiRP configuration file on the remote host.

The network configuration files on your Linux system which you need to modify or check are:

- /etc/rc.d/rc.inet1 or /etc/rc.net, the network configuration script (it may be called something slightly different, depending on your distribution)
- /etc/hosts, the hostname-to-address configuration file
- /etc/resolv.conf, the resolver configuration file

Configuration of these files is discussed in detail in the README.linux file that comes with pppd under the heading “General Network Configuration”; however, a few items are important for using pppd with SLiRP:

- In /etc/hosts, use the address `10.0.2.15' as the address for your Linux machine; this is the address SLiRP uses by default. For example, my own machine is called “zephyr”, in the fictitious domain “earth”, and the following line appears in my host table after the loopback entry:

```
10.0.2.15 zephyr.earth zephyr
```

- In order to fill in the **nameserver** part of /etc/resolv.conf, you need to know the address of the nameserver for the remote host. There are three ways to find out this information, in order from least to most effort:

  (1) Use the command **nslookup**, which will print the name and numeric address of the default nameserver (use **exit** to exit nslookup);

  (2) Read the remote resolver configuration file using the command **cat /etc/resolv.conf**--it should have nameserver entries just like the ones you need; or

(3) Ask the system administrator, who may or may not be willing to give you that information.

You may wish to perform certain actions when the PPP link opens and to perform others when the PPP link closes down; you can put such actions in scripts called /etc/ppp/ip-up and /etc/ppp/ip-down. For instance, I have configured smail (my mail delivery agent) to send outgoing mail to my Internet account provider, which is only available when my PPP link is up. In my ip-up script, I have a command to send any queued outgoing mail along to the smart-host for delivery.

```
#!/bin/sh
#
# /etc/ppp/ip-up -- do net-stuff when ppp is up
# send queued outgoing mail over new net connection
/usr/sbin/runq
```

The ip-up and ip-down programs do not have to be shell scripts—they could just as easily be written in Perl, Tcl, or any other scripting language. However, the scripts should be executable, i.e. they should have execute permission set (using

Do not break up the following line:

```
chmod 0755 /etc/ppp/ip-up /etc/ppp/ip-down
```

and the first line of the script should be of the format **#!*path-to-program***, containing the complete path to the program that should run the script (e.g., **#!/bin/sh** for a Bourne shell script, or **#!/usr/local/bin/perl** for Perl).

pppd sends messages and some debugging information to the system logging facility **syslogd**. You may wish to configure syslogd so that those messages get logged to a separate log file; to do this you need to become the superuser and modify the syslogd configuration file /etc/syslog.conf. pppd logs to the “daemon” facility, so we add the following line to syslog.conf:

```
# Log daemon-related messages in a special place
daemon.*        /var/log/daemon.log
```

You should put the daemon log in the same place as your other system logs; the Linux Filesystem Standard recommends /var/log. In order to get syslogd to re-read its configuration file, send it a hangup signal using the command **kill -HUP *pid***, where ***pid*** is the numeric process id for syslogd as shown in a listing made by a **ps ax** command. Alternately, use **killall -HUP syslogd**.

Since it can be a bit tedious trying to start and stop pppd from the command line, I recommend creating two executable scripts called ppp-on and ppp-off. The pppd package comes with sample scripts in pppd-2.1.2d/chat, which you

may wish to use as a guide. Simple versions of each are shown in **Figure 1** and **Figure 2**. You should probably install the scripts in the same place as you install pppd (for my system, /usr/local/ppp/bin).

Figure 1. Example ppp-on Script

Figure 2. Example ppp-off Script

Making the Relationship Work

With SLiRP installed on your remote host and pppd installed on your Linux machine, it's time to try to get them to work together. It's a good idea to dial into your Internet service provider (the remote machine) using your favorite communications package (such as kermit, minicom, or Seyon) for the first few times in order to test the login and connection process. There are several steps to take:

- Display the system routing table using the command **netstat -rn** on your local machine. If netstat is not available, try using the command **/sbin/route**. If neither of those commands is available, you may need to install a networking package from your Linux distribution—stop here and check your distribution documentation or consult your local Linux guru. Your routing table should look similar to Figure 3.

Figure 3. Initial Routing Table

- Start your favorite communications package.
- Set your modem not to hang up when DTR is dropped (consult your modem documentation for how to do this—for Hayes-compatible modems, the command is usually **at&d0**).
- If your communications package has a session logging feature (e.g., kermit's **log session** command), turn on session logging.
- Dial the remote machine and log in as you normally do. If your communications package doesn't have a session logging feature, you should record the login procedure by hand, writing down both what prompts appear and what you type in response.
- Start SLiRP from your shell prompt, using the command **slirp**.
- Stop session logging if you started it above, and either suspend your communications program or disconnect it from the modem (I recommend the second; in kermit you can say **set line** at the command prompt to disconnect without hanging up or exiting kermit).
- Start pppd using the command **ppp-on**.
- Wait a few seconds for the connection to come up, and then display the network routing table again. Your routing table should now have entries

for the remote host similar to Figure 4, where ***xxx.xxx.xxx.xxx*** is the IP address of the remote host.

Figure 4. Final Routing Table

- If your routing table appears to be correct, try connecting to the remote machine with the command **telnet** ***xxx.xxx.xxx.xxx***, using the IP address shown in the routing table, and log in as you normally would. If that works, try [telnet *hostname of the remote machine* or telnet to another host that you know of. You're connected! When you're done, use **ppp-off** to stop pppd, and resume your communications program. To stop SLiRP, wait a moment and type **0** (zero) five times. SLiRP actually requires a short pause in between each 0 to recognize it as a shutdown string, but normal typing speed works fine. Log out, and hang up your modem.
- If those routing table entries do not appear, first check that your pppd configuration file /etc/ppp/options contains a **defaultroute** entry. Next, check that there is no entry for **passive** or **silent** in your pppd configuration file—by default, SLiRP silently waits for the other end of the PPP connection (in this case, pppd) to send configuration packets. If the entries are not there, check that pppd is still active, using the command **ps ax | grep pppd**--if it's not in passive or silent mode, pppd will time out if it receives no answer after sending a certain number (10 by default) of configuration requests.
- If pppd times out, check the system logs for entries from pppd. If you didn't configure syslogd to catch messages from pppd as described above, now would be a good time to do that. Note that you can increase the level of system log messages from pppd by putting a **debug** entry in pppd's configuration file. SLiRP will also perform several levels of logging: **slirp -d ~/slirp.log** will log debugging output to the file ~/slirp.log on the remote host; adding the entry **log start** to the SLiRP configuration file ~/.slirprc will log startup information to ~/.slirp_start; and adding the entry **debug** to the PPP section of the SLiRP configuration file will log PPP debugging output to the file ~/ppplog.

Gluing the Pieces Together

Once you have pppd and SLiRP talking to each other, you can create a more sophisticated ppp-on script to automatically dial the remote host, log you in, start slirp, and then start pppd on your Linux machine. I have created such a script which uses the “chat” utility that came with the pppd package; it is shown in **Figure 5**.

Figure 5. Sophisticated ppp-on Script With chat Options

This shell script is designed to look in a certain directory (such as ~/.ppp or /usr/local/ppp/script) for “chat scripts” to use in dialing and logging in. When started with an argument, e.g., **ppp-on *my-isp-name***, this version of ppp-on will use a chat script called ~/.ppp/*my-isp-name*.chat for dialing and logging in. Such a feature allows a single user to have multiple dialing scripts for multiple Internet Service Providers, and allows multiple users to keep their passwords private.

When started with the **-c** option, this ppp-on script can also execute a custom script after starting the PPP connection. For example, **ppp-on -c** would perform a script called ~/.ppp/default.ppp after starting pppd, and **ppp-on -c *my-isp-name*** would dial and automatically log me in using the chat script as described above, then perform **~/.ppp/*my-isp-name*.ppp** after starting pppd. This can be a useful substitute for the /etc/ppp/ip-up script in a multi-user environment. Additionally, this ppp-on script allows a verbose option **-v** to log the execution of the chat script to the system logs for debugging.

A chat script consists of a sequence of “expect-send” strings separated by whitespace. chat expects to receive on its standard input the first non-keyword string in the script; when it does, it sends the next non-keyword string followed by a carriage return to the standard output, and so on. To make a customized chat script, you can use the session log you made above while logging into your remote host. A sample session log and the resulting chat script are shown here; you may wish to consult chat's manual page as well.

First, the sample session log:

```
at
OK
atz
OK
at&b0s0=0
OK
atdtMY-PHONE-NUMBER
CARRIER 28800
PROTOCOL: LAP-M
CONNECT 115200
Welcome to My Internet Service Provider
suchandsuch login: MY-LOGIN-ID
Password:
--SYSTEM NEWS--
        blah blah blah blah
        blah blah blah blah blah blah.
TERM = unknown
TERM = (vt100) vt102
MY-SHELL-PROMPT% slirp
```

And this is the chat script that matches the session log:

```
ABORT BUSY
ABORT "NO CARRIER"
ABORT "NO DIALTONE"
TIMEOUT 2   OK-at-OK atz
TIMEOUT 5   OK at&b0s0=0
            OK atdtMY-PHONE-NUMBER
TIMEOUT 60  CONNECT ""
```

```
TIMEOUT 10  ogin:--ogin: MY-LOGIN-ID
            ssword: \qMY-PASSWORD
            (vt100) vt102
            MY-SHELL-PROMPT% slirp
```

The script uses several features of chat which make it easier to perform automated logins:

- The **ABORT** keyword, which aborts the script at any time if any of the strings listed is received.
- The **TIMEOUT** keyword, which adjusts the chat timeout in seconds, so you don't have to wait for 45 seconds to find out that your modem is not responding.
- The “empty” send string surrounded by quotes, **""**, (chat accepts single or double quotes), which simply sends a carriage return.
- The “sub-expect” sequences using hyphens (if the first string is not received, chat sends the string following the first hyphen, followed by a carriage return, and then expects the string following the second hyphen; if the first string is received, the rest of the hyphenated sequence is skipped)
- The **\q** (“quiet”) escape sequence, which keeps the string from being logged, making sure that your remote password does not appear in the system logs.

If we can make it easier to log into a remote host using SLiRP, pppd, and chat, we ought to be able to make it easier to log out as well. The ppp-off script (see **Figure 2**) does some of this, but not quite enough. pppd does have a **disconnect** option, but since you may wish to use pppd to connect to multiple Internet service providers, one or more of which may not use SLiRP, there's a more flexible solution: instead of trying to use pppd to hang up the phone (which is not its job), we'll use chat to exit SLiRP and hang up the phone after pppd has disconnected the PPP link.

Figure 6. The remote-slirp-off script

Figure 6 contains a shell script called remote-slirp-off, which does just that. This shell script shows chat scripts included as arguments to chat instead of read from a script file. Note the use of double-backslashes for chat escape sequences—if they weren't doubled, the shell would remove them, and the characters which follow would simply appear to be part of the send string. The **\d** escape delays for one second, while the **\p** escape delays for a tenth of a second. **\c** indicates not to send a carriage return after the send string in which it appears. The script also provides a verbose option **-v** to log the chats to the system logs for debugging. Like chat, remote-slirp-off expects the modem to be on both standard input and standard output.

Armed with the “glue” of the foregoing scripts, including your custom-made chat script for dialing into your Internet account, you can start a PPP connection between pppd and SLiRP using the simple command:

```
ppp-on name-of-chat-script
```

and you can stop the connection using the simple command sequence:

```
ppp-off
remote-slirp-ff </dev/modem >/dev/modem
```

If you simply must have simple, one-word commands to perform these actions, you can easily put them into an alias, shell function, or yet another shell script.

Epilogue

In addition to the features and capabilities I have discussed here, SLiRP has more sophisticated facilities for load-balancing and restarting interrupted connections, as well as utility programs for starting a remote shell without having to log in via telnet or for sending a file over a SLiRP connection without having to use FTP; however, none of these are necessary for a working Internet connection. Check out the documentation accompanying SLiRP and the SLiRP WWW pages for more information about those extras.

Be careful not to get too addicted to a SLiRP connection. Remember: it's like having a telephone, only more expensive.

Jim Knoble (jmknoble@mercury.interpath.com) has been friends with computers since the days of homebuilt Heathkit Z80 machines running CP/M and has been learning about Linux since late 1992. When Jim's not at work, he enjoys international folk dancing, singing in his church choir, brushing up on his German, cycling, baking bread, hiking, and writing poetry. He lives in Chapel Hill, North Carolina, home to many fine Linux products and personalities. Jim hopes that one day the world will experience lasting peace.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

Building a Linux firewall

Chris Kostick

Issue #24, April 1996

Learn about the three types of firewalls—application proxy gateway, circuit level relay, and packet filter—and how they are used to protect your network from unauthorized access.

The growth of the Internet has prompted many organizations to become security-conscious. Documented and undocumented incidents of security violations, expanded research about security issues, and even media hype have brought about the potential for at least partial solutions for securing a networked environment—without completely isolating the network from the outside world. Leading the pack of solutions is the firewall. Just about everyone has defined what a firewall is, so I won't be any different. A firewall is a device or collection of devices that restricts the access of “outside” networks to “inside” networks. Not surprisingly, Linux can play a part in this arena.

There are currently three models used to classify firewalls. Fundamentally, the current industry classifications are application proxy gateway, circuit level relay, and packet filter.

An application proxy gateway is what most people think of when they talk about firewalls. Also known as a bastion host, it is used to completely sever the connectivity between outside and inside networks. Connections are made via proxy processes to the bastion host. The bastion host in turn will establish a connection to the real destination and handle communications between the two connections.

There are several advantages to a proxy gateway. First, because the proxies are at the application level, they can take advantage of application protocols. For example, protocols providing authentication—such as TELNET, FTP, and HTTP—can be intercepted at the proxy and stronger authentication mechanisms applied (such as S/Key) without adversely affecting the remote client. Also, protocol-specific rules can be applied by the proxy. A rule can be established

that allows FTP **GET**s through the gateway, but not FTP **PUT**s. Another advantage is the extensive logging that can be provided at the application level. It is important to note that the bastion performs no IP routing functions. All communications are through proxy processes. The firewall toolkit **FWTK**, available as freeware from TIS, is an example of a firewall application level gateway.

A circuit-level relay functions in a manner similar to an application proxy gateway, except the proxies employed for a circuit relay are normally not application-aware. Because of this, you lose many of the detailed logging capabilities and precise rule definitions you have in an application proxy gateway. The important concept remains the same in that a connection is established via proxies and IP packets are not forwarded through the firewall. **SOCKS** is an implementation of a circuit level relay based firewall.

Packet filtering is the most common type of firewall available. It works on the concept of forwarding packets based on rules. Those rules typically take into consideration source and destination IP address, source and destination port numbers, the protocol being transported, TCP flags, IP flags or options, and other information, such as the interface, over which the packet arrived. The primary difference between a packet filtering firewall and the others is IP forwarding. A packet filtering firewall is usually a router, and its function in life is to forward packets. This means that while you can control what machines on the outside can talk to certain machines on the inside (and which applications), you now rely on the application to protect itself from harm. For some applications, this isn't a particularly wise decision. Nonetheless, packet filtering can be very useful, is widely available and typically inexpensive.

A Linux machine can function as any one or as all three (i.e., as a hybrid) of these firewall types) of the firewall types. Without add-ons however, the Linux kernel has the ability to function as a packet filter routing device, using the ipfirewall code written by Daniel Boulet and Ugen J.S. Antsilevich. For most 1.2.x and 1.3.x kernels, the firewall code (ip_fw.c) is based on the port by Alan Cox and Jos Vos. Boulet has released version 2.0e (as of this writing) of his ipfirewall code as shareware. I have yet to install the new release, so any discussion I have is based on the ip_fw.c code—specifically kernel 1.2.13.

Caution

In order to use this built-in firewall capability, you need to understand a bit about how TCP/IP works. Trying to set up a firewall from scratch without understanding networking is a sure route to disaster. If you want a “plug and play” firewall solution for Linux, one is mentioned at the end of this article. To learn more about TCP and IP, recommended reading is *TCP/IP Illustrated,*

Volume 1 by W. Richard Stevens. Also, the 3rd edition of Douglas Comer's *Internetworking with TCP/IP* is excellent bedtime reading.

How It's Done

The firewall code includes three facilities—accounting, blocking, and forwarding. *Accounting* rules are used for maintaining packet and byte count statistics for selected IP traffic. *Blocking* rules specify rules for accepting and rejecting packets to and from the firewall itself. *Forwarding* rules specify which packets will be forwarded by the firewall; this implies a source and destination address of something other than the firewall. You can specify any type of rule based on source and/or destination IP addresses; TCP or UDP ports; protocols such as TCP, UDP, or ICMP; or by combinations of the three.

The services are activated when the kernel boots, and the rules are set and modified with the **setsockopt(2)** system call. The current accounting statistics and firewall rules can be viewed by looking at the files ip_acct, ip_block, and ip_forward in the /proc/net directory. The contents of ip_acct look like this:

```
# cat /proc/net/ip_acct
IP accounting rules
C0A80101/FFFFFFFF->00000000/00000000 00000000
 0 0 0 386 392946 0 0 0 0 0 0 0 0 0 0
```

In this example, one rule is present, which basically says to keep statistics on connections from 192.168.1.1 to anywhere for all ports and all protocols.

Changing the accounting and firewall facilities has to be done via a C program, Perl script, or some other language that supports the **setsockopt(2)** system call. Here is a sample program that will change the default policy for the forwarding rules:

```
# cat set_policy.c
#include <stdio.h>
#include <string.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/ip_fw.h>

main(int argc, char **argv)
{
  int    p, sfd;
  struct ip_fw fw;

  fw.fw_flg = 0;

  if (strcmp(argv[1], "accept") == 0) {
    p = IP_FW_F_ACCEPT;
  }
  else if (strcmp(argv[1], "reject") == 0) {
    p = IP_FW_F_ICMPRPL;
  }
```

```
    else if (strcmp(argv[1], "deny") == 0) {
      p = 0;
    }

    sfd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    setsockopt(sfd, IPPROTO_IP, IP_FW_POLICY_FWD,
               &p, sizeof(int));
  }

# cat /proc/net/ip_forward
IP firewall forward rules, default 4

# set_policy deny

# cat /proc/net/ip_forward
IP firewall forward rules, default 0
```

As you can see, it is just a matter of opening up a raw IP socket and using **setsockopt()** to change the environment. The setsockopt call is setting the default policy for the forwarding rules (**IP_FW_POLICY_FWD**). The value for the forward policy command is kept in **p** and determined by the command-line arguments **accept**, **reject**, and **deny**. The words are equated to the defined values of **IP_FW_F_ACCEPT**, **IP_FW_F_ICMPRPL**, and **0**. The difference between deny and reject is that the deny policy will just throw packets away, while the reject policy will throw packets away, and also respond with an “ICMP port unreachable” message to the originating host.

Don't Reinvent The Wheel

Writing the C or Perl code to manipulate firewall rules may sound like a lot of fun, but some administrators may not have the time to roll their own firewall interface. **ipfw** is a utility in the net-tools (version 1.3.6) package that will allow the root user to add, delete, or list information dealing with the accounting and firewall rules. Figure 1 shows the output of the blocking rules in the current host.

Obviously, this is a lot better than looking at the direct contents of the /proc/net/ip_block file. The **-n** option just tells ipfw not to resolve addresses to names.

Adding a rule is done quite easily. Let's say we wanted to accept SNMP queries from a remote management station (note that the author does not really advocate this kind of behavior—this is just an example). We could add the rule:

```
# ipfw add b accept udp from 0.0.0.0/0 \
  to 20.2.51.105 161
```

to give us the new list shown in Figure 2.

Before we move on, let me explain some of the notation in the ipfw command given above. The arguments tell us we're adding a blocking (**b**) rule for the UDP protocol. The rule is accepting UDP datagrams from any host (0.0.0.0/0) to

20.2.51.105, but only those destined for port 161. 20.2.51.105 is the address for one of the interfaces on the firewall.

Assume the above rule for SNMP is not what we wanted. Let's say we only wanted to allow SNMP queries from one particular network—for example 20.2.61.0 (subnetted of course). We can delete the rule we just added and then put in our new rule.

```
# ipfw del b accept udp from 0.0.0.0/0 \
  to 20.2.51.105 161
# ipfw add b accept udp from 20.2.61.0/24 \
  to 20.2.51.105 161
```

Figure 3 shows our new rule set. The syntax 20.2.61.0/24 allows you to specify a netmask with the address. The /24 says there is a 24 bit netmask or 255.255.255.0.

The more astute reader at this point may be asking, “What the heck do those rules mean anyway?” I'll get to that, but first, I want to talk about a utility which I find preferable to ipfw.

The **ipfwadm** (version 1.2) program by Jos Vos (available from www.xos.nl/linux/ipfwadm) is an administrative utility for IP firewalling and accounting which is similar to ipfw. It offers, I think, a slightly more intuitive interface, better output, and a better man page (not everyone reads the source code for documentation).

To list the rules shown in Figure 3, issue this command:

```
# ipfwadm -B -l -n
IP firewall block rules, default policy: accept
typ prot source         destination   ports
den tcp  0.0.0.0/0      20.2.51.105   * -> *
den tcp  0.0.0.0/0      192.168.1.1   * -> *
acc udp  20.2.61.0/24   20.2.51.105   * -> 161
rej udp  0.0.0.0/0      192.168.1.1   * -> *
acc udp  0.0.0.0/0      20.2.51.105   53 -> *
rej udp  0.0.0.0/0      20.2.51.105   * -> *
```

Notice a couple of things. First, ipfwadm always shows the default policy; I like to see this. Second, the type fields for these rules:

```
rej udp  0.0.0.0/0      192.168.1.1   * -> *
rej udp  0.0.0.0/0      20.2.51.105   * -> *
```

are set for reject, rather than deny, as shown in ipfw's output in Figure 3. Well, they really are set to reject. ipfw only supports the deny and accept policies, not reject.

Given our vast knowledge on setting and listing rules, let's rebuild the table again (except for the SNMP rule) while adding a few more details. But first, we'll define what our network looks like, shown in Figure 4.

We'll call the 20.2.51.0 network the “outside” network and the 192.168.1.0 the “inside” network. Since blocking rules apply to the firewall itself, the rules will be set on deathstar. First, we'll flush any rules we have and set the default policy to accept:

```
# ipfwadm -B -f
# ipfwadm -B -p accept
```

Now we'll define what we need to block. The protocols you can choose to block are TCP, UDP, and ICMP. We want to allow ICMP messages to the firewall, so we can't just block everything. We could block TCP by adding the rule:

```
# ipfwadm -B -a deny -P tcp -S 0.0.0.0/0 \
  -D 20.2.51.105
```

but that would be not be adequate. That rule would end up blocking all traffic **from** the firewall, as well as traffic **to** the firewall. Therefore, an administrator could not telnet or ftp from the machine. That may be desirable, but let's assume we'll let out TCP traffic originating from inside. What we would like is to block all connection attempts to the firewall while letting connections go out. We can modify the rule with the **-y** option. This will do what we want by blocking any TCP segments from any host to the firewall that have the SYN bit set.

```
# ipfwadm -B -y -a deny -P tcp -S 0.0.0.0/0 \
  -D 20.2.51.105
```

Remembering that the firewall has two interfaces, we block the second interface also:

```
# ipfwadm -B -y -a deny -P tcp -S 0.0.0.0/0 \
  -D 192.168.1.1
```

That is too restrictive, in that connections from the inside network to the firewall will be blocked also. We can refine the rule to block all TCP connection requests only if they come in over the outside interface (20.2.51.105).

```
# ipfwadm -B -y -a deny -P tcp -S 0.0.0.0/0 \
  -D 20.2.51.105 -I 20.2.51.105
# ipfwadm -B -y -a deny -P tcp -S 0.0.0.0/0 \
  -D 192.168.1.1 -I 20.2.51.105
```

There, that wasn't too bad.

Now that we've taken care of TCP traffic, let's write some rules for UDP. Since UDP has many problems that we won't discuss in detail here, we'll just block it all. The rules will be the same for TCP, except since there's no SYN bit, there's no need for **-y**, and so we'll reject the packet instead of denying it. The reason we reject it is that ICMP port unreachable messages make sense to UDP based applications, but are ignored by TCP applications. It would be nice if the ip_fw code sent TCP Resets if the rule was for TCP and marked for rejection, but it doesn't. So our rules for UDP will be:

```
# ipfwadm -B -a reject -P udp -S 0.0.0.0/0 \
  -D 192.168.1.1 -I 20.2.51.105
# ipfwadm -B -a reject -P udp -S 0.0.0.0/0 \
  -D 20.2.51.105 -I 20.2.51.105
```

Setting up those rules, we find that blocking all UDP traffic isn't such a good idea after all. If we have telnet, then we will probably want to be able to resolve hostnames. So we open up DNS. Before we do that, though, let's look at the traffic pattern for a DNS query so that we can gauge what we'll need to write. Figure 5 contains tcpdump output of a DNS query from deathstar to mccoy, an “outside” DNS server.

We can see we will have to have two rules—one for the query going out (sending to port 53 on the remote machine) and one for the response (sending to port 53 on the firewall from the remote machine). The rule can be written as:

```
# ipfwadm -B -a accept -P udp -S 20.2.51.105  \
  -D 0.0.0.0/0 53
# ipfwadm -B -a accept -P udp -S 0.0.0.0/0 53 \
  -D 20.2.51.105
```

Since this two-way traffic is common among many protocols there's an option you can set to condense the two rules into one. The **-b** option sets bi-directional mode, which installs a rule that matches traffic in both directions. We can then get away with

```
# ipfwadm -B -a accept -b -P udp -S 0.0.0.0/0 53 \
  -D 20.2.51.105
```

Now that we've recreated our table, we can get a listing of the rules with extended (**-e**) output. This is shown in Figure 6. Notice the extended output contains packet and byte counts for the blocking rules. The firewall code automatically maintains accounting information for the forwarding and blocking rules.

What we have accomplished so far is protecting the machine deathstar to suit our environment. To protect the internal network we will need to develop the proper forwarding rules. Before I just start typing in rulesets, I like to build a table of what's allowed and disallowed. Figure 7 shows the table I put together

to establish the blocking rules. Note that the asterisk in the rules table indicates any host address or any port number.

We can build the same type of table for our forwarding ruleset. Building our table should be simple if we have a security policy in place. Let's assume that the part of our security policy that discusses firewalls will allow TELNET and FTP out, and e-mail (SMTP) in both directions. Further, e-mail is only allowed to go to the mailhub (since there's only one machine on the internal network, it will be the mailhub—humor me). Figure 8 shows the generic rules table for the forwarding ruleset.

The stance of the firewall is one of “deny everything”. This is a very common policy for firewalls because the only packets that will be forwarded are the ones which are explicitly allowed.

The rules are fairly straightforward except for the source and destination ports numbered 1024 or greater. The reasons for this are basically historical. Most Unix client programs, such as TELNET, assign their ephemeral port range between 1024 and 5000. Ports 1 through 1023 are known as “reserved ports”. These are for server applications such as telnetd, ftpd, etc. Almost all TCP/IP stacks follow this convention and because of it we can take advantage of it in our ruleset—to help guarantee client-only communications coming into the network. The reason I don't use the range 1024-5000 is because not all devices adhere to this Unix tradition. For example, annex terminal servers start their ephemeral port range at 10000.

Here are the commands to establish our forwarding rules:

```
# ipfwadm -F -f
# ipfwadm -F -p deny
# ipfwadm -F -a accept -b -P tcp -S 0.0.0.0/0 23 \
  -D 192.168.1.0/24 1024:65535
# ipfwadm -F -a accept -b -P tcp -S 0.0.0.0/0 21 \
  -D 192.168.1.0/24 1024:65535
# ipfwadm -F -a accept -b -P tcp -S >0.0.0.0/0 20 \
  -D 192.168.1.0/24 1024:65535
# ipfwadm -F -a accept -b -P tcp -S >0.0.0.0/0 25 \
  -D 192.168.1.2 1024:65535
# ipfwadm -F -a accept -b -P tcp -S >192.168.1.2 25 \
  -D 0.0.0.0/0 1024:65535
```

To set up deathstar as the firewall machine, the ipfwadm commands would be put into a file and executed as a shell script. Deathstar uses the file /usr/local/etc/set-rules.sh. To bring the machine up properly, it would be wise to establish the rules within the kernel before the network interfaces are brought up. The /etc/rc.d/rc.inet1 on deathstar contains the lines:

```
# set firewall rules
/bin/bash /usr/local/etc/set-rules.sh
# bring up ethernet
ifconfig eth0 192.168.1.1 192.168.1.255 up
```

```
# bring up ppp link
/usr/lib/ppp/ppp-on
```

Deathstar is, in reality, my desktop machine. I've loaded just about everything on it, so it doesn't make a very good firewall. A firewall, as we have described it, should run the bare minimum of software in order to function. Normally this means that compilers, X, games, or anything that doesn't have to do with the kernel or communications are taken off of the system.

Checking Your Work

Even with generic tables to work with, you may not always get the rules the way you want them. It's nice to be able to check your work. The ipfwadm utility offers the **-c** option to check packets against your rules. For example, to check if a packet from some host can send mail to an internal host other than warbird, we can run:

```
# ipfwadm -F -c -P tcp -S >195.1.1.1 1024 \
  -D 192.168.1.5 25 -I 20.2.51.105
```

This would yield the response **packet denied**. When using **-c** to check a rule, you need to be very specific and supply a source address and port, destination address and port, and an interface address.

The other way to test your environment is with live traffic. If you suspect traffic is not being forwarded because of your ruleset, you can use tcpdump to monitor the traffic coming into and going out of the firewall. It becomes fairly obvious if the firewall is not allowing legitimate traffic to go through. For example, when I set up the rules to allow mail through, I noticed it took an exceptionally long time to send a message. tcpdump revealed that the receiver, mccoy in this case, was sending IDENT messages back to the source but they were being blocked by the firewall. By adding a rule to allow IDENT messages, mail went much faster. Creating this rule is left as an exercise for the reader.

For rudimentary logging, a rule may be set with the **-k** option, which will cause the kernel to print out a message via syslog for all matching packets. However, setting up the kernel to understand the **-k** option is not straightforward. The kernel needs to be compiled with **CONFIG_IP_FIREWALL_VERBOSE** defined. To do this, just add the definition to the Makefile in the net/inet directory in the kernel source directory. Unfortunately, the code defined in the **CONFIG_IP_FIREWALL_VERBOSE** section of ip_fw.c does not compile cleanly in 1.2.x distributions. The fix is simple and implemented in the latest 1.3.x versions of the kernel.

If you set up the kernel to support the **-k** option you will receive output in the /var/adm/messages file similar to that shown in Figure 9.

Concluding Comments

The firewall we just built can be replaced by almost any router you can purchase from a vendor. However, turning a Linux machine into a packet filtering router is a cheap and very effective alternative.

There are several limitations to the firewall code. Its inadequate logging capabilities are a big miss; documentation is lacking; and the inability to filter on IP options do not allow the filtering router to be as flexible as it might be.

For many environments, the firewall facilities of the Linux kernel can be more than sufficient, but for those who need commercial-grade firewall software for Linux, or software that can run under Linux, there are solutions. The shareware ipfirewall code from Daniel Boulet, mentioned earlier in this article, addresses several of the problems just stated. Also, the commercial Mazama Packet Filter from Mazama Software Labs is a real “bells and whistles” product. It comes complete with nice documentation, a filter “language” for defining the rulesets (this is a winner), a GUI for very simple administration, and fixes for the technical problems (such as IP options and TCP SYN/ACK filtering).

One last concept not mentioned in this article is that of IP Masquerading. Very perceptive readers will notice the network warbird is on (192.168.1.0) is a private IP address. That is, it not one assigned by the InterNIC, but can be used for local or private IP-based networks not connected to the Internet. I can get away with using this addressing scheme because the machine called relay is a commercial firewall that performs masquerading (otherwise known as address hiding). All connections going out of the 20.2.51 or 192.168.1 networks have a source address of relay from the perspective of the remote machine. As you might be able to guess from its name, relay is also an application proxy gateway. Linux also has the ability to hide addresses, but that is a topic for another article.

Chris Kostick (ckostick@csc.com) is a Senior Computer Scientist at Computer Sciences Corporation's Network Security Department. He enjoys working with Linux but considers himself a latecomer because he started out at kernel version 1.1.18. As far as computers go, he's not sure if he has more fun debugging TCP/IP problems or playing Doom.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

XF-Mail

John M. Fisk

Issue #24, April 1996

If you have a dial-up connection to the Internet, one of your easiest options for getting and receiving e-mail is popmail. But to use popmail, you need a “popmail client”. XF-Mail is a convenient, easy-to-use, and reasonably powerful popmail client that is worth a closer look.

I suspect that there are a **lot** of folks who are in a position similar to mine: using Unix at school or work on a minicomputer or mainframe, and running Linux on our PCs at home. If this is the case, it is quite likely that your Internet connection is via a dial-up connection, PPP, CSLIP, or SLIP. If this is the case, your options for getting mail into and out of your Linux box have almost invariably included having to set up sendmail, sendmail+IDA, or smail. Setting any of these up is not a trivial task. If you are fortunate, your choices also include **popmail**, in which your mail is left on a *popmail server* in your own mailbox, waiting for you to retrieve it at your leisure. But to retrieve it, you need a mail program installed on your Linux box which can talk to the popmail server.

XF-Mail To the Rescue.

What I'd like to do is take an in-depth look at compiling and configuring XF-Mail. Let me offer one caveat at this point: this is **not** meant to insult anyone's intelligence. I'm writing this primarily for the benefit of those DOS-converts, like myself, who may initially view the notion of compiling programs as a task for wizards. It's not that bad. Honest! However, until you get the hang of this, getting Linux programs working may well be a formidable task. If you're pretty adept at compiling programs, you may want to skip to the next section, on configuration, although even that is admittedly pretty straightforward.

Just a word for those of you who are left. If you are an absolute Linux newbie and the thought of compiling anything sorta gives you the heebie-jeebies, stick with me. This will be fun. Really!

The philosophy behind Linux, and much of Unix, is vastly different from that of DOS. While you can often find pre-compiled binaries that work simply by dropping them into a directory in your path, there are MANY instances in which this is simply not the case. You'll get an archive with the .tar.gz suffix and un-archive it, only to discover that it's all source code and needs to be compiled. Eventually, you'll want to be able to do this. It's often not as hard as it first seems, although admittedly, not everything compiles cleanly “out of the box”.

I'm going to take you through each step needed to compile XF-Mail. I think this is extremely important, for a couple reasons:

- This will be good practice in compiling and setting up a couple programs that are quite well behaved and need little modification to get working.
- If you're using Linux on a home PC and have a dial-up connection, you really need to have an easy-to-use e-mail client. You need to be connected. Seriously. You're quite handicapped and cut off if you don't.
- If you've never compiled anything under Linux, this is your golden opportunity to do so and feel pretty good about yourself!

Let's begin. You'll need to have a few things set up in order to do this:

- You need to have X-Windows up and running.
- You need to have installed the development applications—for those of you using Slackware, that's the stuff in the D disk sets. Other distributions will use different designations. You should have the GCC (GNU C Compiler) installed, as well as the include files and so forth.
- You need to ftp some files.
- You need to be able to use at least one of the Linux text editors such as vi, Emacs, xedit (if you're running X-Windows), pico, joe, jed, or one of the many others.

You need to get the following files (use a web browser):

- xfmail-0.3.tar.gz (Burka.NetVision.net.il/xfmail/xfmail.html)
- bxform-075.tgz (http://bragg.phys.uwm.edu/xforms/)
- libXpm-3.4f.tar.gz (ftp://ftp.cc.gatech.edu/pub/linux/libs/)

The first two links—those for xfmail-0.3.tar.gz and bxform-075.tgz—will take you to the home pages of these impressive programs. Read the info and follow the directions for getting the Linux version of these programs. The last link—for the libXpm stuff—will drop you off at one of the sunsite mirrors (that's GA Tech, if you're curious about this). Because versions of programs can change fairly quickly, I'll leave it up to you to get the most recent version. I'm currently using

the libXpm-3.4f.tar.gz sources. Note that you may already have libXpm installed, in which case you don't need to install it again.

```
$ ls -l /usr/X11R6/lib/libXpm.so*
```

will tell you whether you have the library available.

In case you are curious, those files you got contain:

```
xfmail-0.3.tar.gz
```

The source code for the XF-Mail client version 0.3.

```
bxform-075.tgz
```

The binaries for the XForm toolkit libraries.

```
libXpm-3.4f.tar.gz
```

The source code and precompiled libraries for the libXpm libraries.

XF-Mail needs those last two libraries in order to be compiled. As an additional benefit, there are lots of programs that require the libXpm libraries, and a growing number of programs use the XForm libraries. Having set these up once, you'll be all set when it comes time to compile other programs that require them.

Since I'm using Slackware 2.2.0 at the moment, my descriptions of where things will go may be just a bit different from yours, if you are using a different distribution. Not to fret... I'll try to be as broad as I can in terms of describing how to proceed.

Let's begin by setting up the libXpm libraries. If you already have them, skip this section. If you use one of the new ELF-based distributions, you probably already have them, but if you don't, you will have to get ELF versions instead of the a.out ones documented here. The procedure is the same.

Since the libXpm-3.4f.tar.gz archive contains precompiled libraries and the necessary header files, let's use these. On my system, I have created a /usr/local/src directory in which I store and compile all new programs. You don't have to do this yourself, but it does help keep things organized. Now, let me quickly take you through the steps to set this stuff up:

```
$ mv libXpm-3.4f.tar.gz /usr/local/src/
$ cd /usr/local/src
$ tar -xvzf libXpm-3.4f.tar.gz
$ cd xpm-3.4f/
$ ls -l
```

look for libXpm.a, libXpm.sa, and libXpm.so.4.6. Now become root, and install the files:

```
$ su (type your root password)
# cp libXpm.* /usr/X11R6/lib/
# cp lib/xpm*.h /usr/X11R6/include/X11/
# ldconfig
# exit
bni:Congrats! The libXpm libraries should now be installed.
Make sure you go back to being a normal user again by pressing
Ctrl-D. No point is doing something dumb as root and making
a mess of things, right?
```

Next, let's deal with bxform-075.tgz. This archive contains the binaries (actually, the static libraries) for the XForm libs. It's almost as easy to set up as the libXpm libs, except you'll need to do a little bit of editing to put things where they belong. Note that if you have an ELF-based distribution, you will need to get the ELF version of the library, rather than the a.out version.

We'll start the same way we did the last time:

```
$ mv bxform-075.tgz /usr/local/src
$ cd /usr/local/src
$ tar -xvzf bxform-075.tgz
$ cd xforms/
$ ls -l
```

One of the files you will notice is a README file. It's always a good idea to read through the README files for anything, and this is no exception. It'll inform you that the file you need to edit in order to make any modifications is the mkconfig.h file. It's another good idea to always make a backup copy of any file you modify before you modify it. If things get screwed up, you should have a “clean” copy to fall back on. I've called my backup copy mkconfig.h.dist since it came with the distribution. **cp mkconfig.h mkconfig.h.dist** does the trick.

There are two modifications you may want to make. When you load up the file, you should see lines a bit down the page that look like this:

```
LIB_DIR=/usr/lib
LIBMODE=644
HEADER_DIR=/usr/include
HEADERMODE=644
```

You might want to consider installing the libforms.a library into the /usr/X11R6/lib directory, and the header file, form.h into the /usr/X11R6/include/Xf directory, because that is where XF-Mail will be looking for these files. If you do this now, it'll be a bit easier during the next step. So, change the **LIB_DIR=** and the **HEADER_DIR=** lines to read:

```
LIB_DIR=/usr/X11R6/lib
LIBMODE=644
HEADER_DIR=/usr/X11R6/include/Xf
HEADERMODE=644
```

This should put everything where it belongs.

Now, all that remains to finish the installation, after you've saved the modifications, is to become root again, and type in **make install**.

You are now ready to compile XF-Mail. At this point you merely un-archive the file, copy a file, optionally edit the Makefile (which orchestrates the process of compiling), and compile the program.

```
$ mv xfmail-0.3.tar.gz /usr/local/src
$ cd /usr/local/src
$ tar -xvzf xfmail-0.3.tar.gz
$ cd xfmail/ui
$ cp Makefile.Linux Makefile
At this point, there's really nothing much left to do but
compile the program. I'll add here that you can optionally
change the directories in which the program and the manual
pages are installed. Why? Well, there are a couple of schools
of thought regarding the installation of new programs. The one
that I ascribe to suggests putting all user-added programs into
the /usr/local directory.
You don't need to do this following step, but to change
the default location of the program installation from
/usr/X11R6 to /usr/local, let's take a look at the Makefile.
When you edit the Makefile you've just created, you'll see a section
near the top that looks like:
```

```
BINDIR = /usr/X11R6/bin
MANDIR = /usr/X11R6/man/man1
```

You'll notice that the section delimited by the **# change the following...** statement at the top and the **# end of user configurable settings** statement at the bottom is the only section of the file that you should edit. Also notice the lines that define the XForm include directory (**XFINC**) and the line for the XForm library directory (**XFLIB**). Now you'll see why we edited the Makefile above for XForms: /usr/X11R6/include/Xf is where XF-Mail will look for the form.h header file, and /usr/X11R6/lib is where it will look for the libforms.a file.

To install the xfmail program into your /usr/local/bin directory, and the manual pages into the /usr/local/man/man1 directory, simply edit the **BINDIR** and the **MANDIR** entries:

```
BINDIR = /usr/local/bin
MANDIR = /usr/local/man/man1
```

At this point, all that's left to do is compile and install the program. Simply enter:

```
$ make
$ su
# make install
# exit
```

The first step will take a while, especially if you don't have very much memory, but if the compilation has gone well you should see no error messages. After the **make install** step, the program will be completely installed.

The hardest part is done.

If you have troubles of any sort compiling XF-Mail, you can take the easy way out. The author has Linux binaries available for ftp at burka.netvision.net.il in /pub/xfmail/.

Configuring XF-Mail

XF-Mail was written by a couple of guys named Gennady B. Sorokopud and Ugen J. S. Antsilevich. In their own word:s

XF-Mail is an X-Windows application for receiving electronic mail. It was created using XForms library toolkit by T.C. Zhao and Mark Overmars.

It's partially compatible with MH style mailboxes but it does not require any mh tools to be installed on the system. You can read most of your MH folders and messages with XF-Mail.

XF-Mail has very friendly user interface and it's extremely easy in use. It implements most of the mail functionality in one program and it does not require any additional tools.

Guess that pretty much sums it up! It really does have a very friendly user interface and is quite easy to use. Additionally, while it can be set up to use sendmail, it can happily live without it.

One of the greatest things about XF-Mail is that all the user-configurable options are easily set using the GUI configuration menu. This is a **huge** plus, as the options in the Configuration Menu give you a great deal of power over how the program functions and appears.

To configure XF-Mail, you simply need to start it up. Fire up X, and in one of the xterms enter:

```
$ xfmail &
```

This will fire up the program. Now, depending on your hardware and software set up, this may take a few seconds.

Parenthetically, it's not a bad idea to use an xterm to start any program up that you've just installed. For example, you could install a program and simply add it

to a button bar or menu and call it from there. The problem with this is that error messages are printed if the program can't run. If you try to start a program up from a menu or button bar, you won't necessarily see these messages and will then be left wondering why your program won't start. If you test drive your newly installed programs using the command line in an xterm, you should see what's happening.

There are, fortunately, only a couple things you'll need to know in order to get XF-Mail up and running. These are:

- The DNS name or the IP address of the Mail Host from which you'll get your mail
- The port number or service of your Mail Host—e.g., **pop3** if your Mail Host uses POP3 protocol
- Your username
- Your password
- The DNS or IP address of your SMTP gateway or host

That's pretty much it, as far as essential information goes. The Mail Host information is simply the host from which you get your mail. The SMTP gateway is simply the host to which you'll send your mail. These are usually the same host. For example, I get my mail from the Mail Host ctrvax.vanderbilt.edu (at Vanderbilt University). The mail server there uses POP3 protocol and I use the same address for outgoing mail using SMTP.

If you have any questions about this information, do yourself a **huge** favor and ask or call the folks who are responsible for your mail service. If you're at a university or large organization, there's often a help desk that can provide you with all the information you'll need. If you have mail service through a local ISP, they should be able to help you as well. Essentially, you'll just need the address (either the name or the numeric IP address) of the Mail Host and the SMTP gateway. Since the mail server at Vanderbilt uses the POP3 (Post Office Protocol) protocol, I use that for the “port number”.

To start configuration, go to the menu bar and select the Misc menu and choose Config. You will be presented with a Configuration Dialog box that lets you set a variety of things. Let's first get the essentials out of the way.

To get mail into and out of your box through the mail host and your SMTP gateway, you'll need to enter this information. From the Configuration menu, choose POP which presents you with the POP Configuration menu box. Here, enter the mail host (your POP server from which you'll get your mail) in the box entitled Host. To the right of this, in the space labelled Port, enter the port number or service. If your mail host uses the POP3 protocol, simply leave this

as **pop3**. If it's something else, you should check with the postmaster at your mail service and make sure you know which service/port they use.

Below this, enter your username and your password. Also, check the option buttons if you wish to leave your mail on the mail server and if you want XF-Mail to store your password in the configuration file so that you won't be prompted for it each time you attempt to get mail.

A Couple Caveats:

This, and the section on setting up the SMTP gateway information, are the most critical aspects of configuring your mail client. If you enter an incorrect address for the mail host or the port number, you won't be able to get mail. Second, keep in mind that storing your password, while very convenient, is also **very insecure**, since the password is stored in your ~/.xfmailrc file un-encrypted. If you're the only one using your system, this is not a real problem, but if others have access to your system, be aware that your password is quite vulnerable. Finally, as a precaution, at least at first you probably should **not** delete your mail from the pop server. That way, if anything goes wrong, you'll at least have a copy of your mail on the mail server and should be able to retrieve it again later. If something goes wrong and your mail has been deleted from the server, it's **gone**.

Did I scare you? I hope not; just be prudent and use your common sense. Keep in mind that this program is still in beta testing, and while it has undergone a lot of testing and debugging so far, it is likely to have a few “undocumented features” left. Play it safe.

Lastly, let's set up the outgoing mail. Press the Done button after you've entered all the info for the POP Configuration, and then presss the Send button, which will allow you to set up the SMTP information. In the first box, entitled “SMTP Host”, enter the name or IP address of your SMTP gateway and then the SMTP port number in the box to the right of this. The default for the port is **smtp**, which is an Internet standard.

If you have installed the sendmail program, you can also fill in its location, but it's not necessary.

To have XF-Mail use the built-in SMTP support, click on the **use SMTP** button. A couple other suggestions and we're almost done. First, I recommend that you also check the “Save to **sent mail**” and the “Offline send” options. The first will save a copy of all your outgoing mail. Why would you want to do that? Good question.

Mail on the Internet **can** get lost. Machines crash, hard drives fail, mail delivery programs have bugs. If you save copies of outgoing mail, you can re-send mail if you find it never made it to its destination.

Setting the Offline send option allows you to compose mail and set it up to be sent out, even when you're not connected!

We're almost done. (Have I said that before?) Press the Done button and when you're back at the Configuration dialog box, press the Save button.

Now, there's just one more thing that you probably should do. Press the Misc button in the Configuration dialog box. This presents you with a list of various options. At the bottom, click on the Open log on startup option. This will log all messages to a “Log Window” which will allow you to see what's happening. Keep an eye on this, especially the first couple times you try to retrieve mail from the server and send mail to the SMTP gateway. Now, to save your changes, press Save once again and you're done.

You'll need to get your SLIP, CSLIP, or PPP connection up and running first. Once it's up, fire up XF-Mail and then go to the **Misc** menu and choose **Retrieve mail**. Keep an eye on the action that's going on in the **Log Window**. This should let you know if everything's set up correctly and XF-Mail was able to make connection with your mail host.

No mail? No problem. Send yourself some!

Up at the top of the screen, click on the **pen** icon on the toolbar. The astute user will notice that moving the cursor onto the icon causes a short description to be printed along the status line at the bottom of the window. Very handy. Now, click on that and you'll be presented with an edit window. Now, it's just a matter of filling in the blanks. At the top of the window you can enter the information for the **From:**, **Subject:**, and **To:** fields.

Now, type in a short test message and when you're done, press the **letter** icon on the top left. If you get an error message about needing to specify at least one recipient, don't forget to press the RETURN key after you've filled in the name of the recipient. If this was done correctly, you'll see the entry move to the box below. You could also use your shiny new **Address Book**. Press the **Address** button and the address book fires up. Enter the name on the entry line and press the Save button. Click on the **To** button and press RETURN. Done.

Almost. Remember that we've set up offline send. To get the message out, you'll need to choose the **Send** menu item, and then click on the **Send all** item. If you have an external modem, you should see the lights lighting up and the

log window should let you know the progress of things. If all went well, you should now have new mail! Go back and retrieve it to see if things are working okay.

Now, let's do just a couple other things before finishing up here. First, there are several things that you'll want to set up. You don't have to, but it'll make using XF-Mail a lot easier and more fun.

The things you'll probably want to do at this point are:

- Set up an external editor
- Set up an external message viewer
- Configure the **From:** field for outgoing messages
- Decide whether you want to additionally retrieve mail from a local mail spool
- Edit your .signature file if you haven't already

You'll want to set up an external editor and viewer because the built-in editor and viewer are quite limited and can handle messages only 30 lines or fewer long. That's not much. However, you can set up any X editor as your viewer/editor. Keep in mind, however, that you probably don't want to use something rather huge, like Xemacs. Granted, this is a great editor, but you'll be able to do three week's worth of wash **and** iron all your socks before it'll finish loading. I initially used xedit, which is admittedly a bit Spartan, but at least it loaded up fast. I'm currently using aXe, which loads up quite nicely but is a **lot** more functional.

So, to set these things up, call the Configuration menu up once again:

- Use the View menu to set up the external viewer. Just enter the command necessary to fire up your editor/viewer—just as you would in an xterm. For example, I'm using the xedit program as my external viewer still. I simply enter **/usr/X11R6/bin/xedit** in the box and then set the number of lines to whatever I want, such as 75.
- Use the Edit menu to set up the external editor. Once again, enter the command line necessary to load up your chosen editor. For example, to use the aXe editor, I've entered:

```
/usr/local/bin/axe -noserver -geometry 74x40+55+0
```

- Use the **Misc** menu to set up the **From:** field. (You'll need to do this in order to correctly set this for outgoing mail.) Remember that most folks who want to write you back simply press the **reply** button and so your

From: field is important in order to get the correct address. Enter your e-mail address on the **From:** field line.

- Use the Receive menu to optionally retrieve mail from a local spool. Enter the directory where your mail spool exists. For example, since I spend most of my time as root, my mail spool directory is /var/spool/mail/root. I have this set up for **both** spool and POP retrieval because I often use the program popclient to retrieve my mail when I'm not running X or when I'm using a background shell program to periodically check my mail. I have popclient set up to deposit mail into my local mail spool directory and so XF-Mail will dutifully pick it up from there as well as from the POP server.

After making these changes, don't forget to press the Save button to save them.

To edit your .signature file, select the Misc menu once again and click on the Signature item. There's an easy-to-use signature editor that pops up and lets you create your .signature file and save it. I found that you cannot use the entire window for your .signature file. If you do, you'll get an error message about the signature being too large. You'll have to keep it to about 8 to 10 lines. Play with it and see for yourself. Internet etiquette has suggested a maximum of 4 lines for a long time, so you may wish to limit yourself to 4 to make the rest of the world happy.

Keep in mind that there are **lots** of nifty configuration options and features left to play with. Let me recommend to you that you skim through the online help system. Just choose Help from the menu bar and choose the Contents item. It will display a help window with all kinds of useful information describing what you can do with this great program. It's time to explore!

Want more? It's easy to join the XF-Mail mailing list: simply choose the Help menu item and click on the Subscribe to mailing list option. If you like this client, go ahead and drop the authors a note of thanks! Their addresses are Gennady B. Sorokopud gena@NetVision.net.il and Ugen J. S. Antsilevich ugen@NetVision.net.il.

John Fisk (XXXXXXXXXXXXXX) After three years as a General Surgery resident and Research Fellow at the Vanderbilt University Medical Center, John decided to “hang up the stethoscope” and pursue a career in Medical Information Management. He's currently a full-time student at the Middle Tennessee State University and hopes to complete a graduate degree in Computer Science before entering a Medical Informatics Fellowship. In his dwindling free time, he and his wife Faith enjoy hiking and camping in Tennessee's beautiful Great Smoky Mountains. An avid Linux fan since his first Slackware 2.0.0 installation a year and a half ago.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Finding Linux Software

Erik Troan

Issue #24, April 1996

There is a treasure trove of software for Linux just waiting for you to download it from the Internet or copy it from a CD-ROM. However, you may need to keep an open mind to recognize a treasure when you see it.

One of the most popular questions asked throughout the comp.os.linux Usenet hierarchy is “Where can I find a program to do *insert your favorite task here*?” About half of these questions result in a three-month thread concerning the merits of WYSIWIG versus markup style text editing while the other half slide into obscurity without any answers at all.

Finding software for Linux on the Internet is pretty easy, but there are three principles about free software you should accept before you start looking:

1) Something close to what you want exists. There are thousands of gigabytes of anonymous ftp space on the Internet and one of them will contain something that you'll consider useful—if you ever find it.

2) Whatever you find won't do everything you need. Most of the free programs you'll soon be wading through were written by one or two programmers whose needs are fully met by the programs they wrote.

3) Whatever you find will do more then you need. Unix software tends to accumulate obscure features at an astonishing rate.

Everyone's annual pilgrimage for an X-based Ami Pro clone illustrates these principles. While GUI-based, somewhat WYSIWIG word processing techniques for Linux do exist, none of them (ez, idoc, TeX/LaTeX/xdvi/gs combination, or a groff/gs combination) will do exactly what you had in mind. Then again, you can balance your checkbook in the middle of ez and and use TeX to get every other character to print upside down.

The first place to look for Linux software is the Linux Software Map (LSM), maintained by Lars Wirzenius (lars.wirzenius@helsinki.fi). It's a collection of lsm entries which are supplied by people who submit files to the primary Linux archives (sunsite.unc.edu and tsx-11.mit.edu). The LSM includes descriptions of packages and full path names to where packages can be found on those two sites.

The easiest way to search the LSM is via the World Wide Web. A number of Web search engines are available, such as siva.cshl.org/lsm/lsm.html. This site provides searching via a searchable index, which is supported by most web browsers. Another web search engine is available at harvest.cs.colorado.edu/brokers/lsm/query.html. This interface is based on forms, and not all browsers support it (though lynx, Mosaic, and Netscape all do). When a match is found, hypertext links enable the user to download the files that matched without leaving the browser.

If you don't have Web access (or if you just like doing things on the command line) you can ftp the latest LSM as one large text file from sunsite.unc.edu:/pub/Linux/docs/LSM.gz. A few Linux-based tools to make searching this file easier are available for ftp from sunsite.unc.edu in the directory /pub/Linux/search.

Each package in the LSM has description and keyword fields which provide an excellent basis for searching. Looking for “sound mixer” via the colorado web site found 15 matches, showing how effective searching the LSM can be.

However, the LSM quickly gets out of date. The filenames and version numbers in it are often wrong, so don't think you're finished. The site names, directory paths, and package names can be used to find the software you're looking for even if the specific filename is wrong. Use ftp to connect to the site and cd to the directory the LSM mentions, and you'll probably find a few different packages that meet your needs.

If searching the LSM fails to find what you need, look at the sunsite.unc.edu and tsx-11.mit.edu ftp sites. Both of those sites have large Linux directory trees (in /pub/Linux and /pub/linux respectively) with both binaries and sources. Most CD-ROM distributors of Linux include a CD-ROM of one or both sites which provide a fast alternative to ftp.

Sunsite provides a file that is especially handy for finding software—/pub/Linux/INDEX.whole.gz. It contains all of sunsite's text INDEX files concatenated together, meaning it has a description of almost all of the files in sunsite's Linux directory. Searching this file for keywords with grep or a text editor lets you skim all of sunsite's 2 gigabytes very quickly. Looking for the word “mixer” in this file found 10 lines that mention the word, one of which is the name of a

directory full of audio mixers! If sunsite has what you're looking for, this method is sure to find it.

Tsx-11.mit.edu's archive doesn't have as nice a way of searching it. It provides two files, ls-lR and find-ls.gz that list all of the files in the system. If you know (or can guess) the name of the file you want, these files will help you find it.

If none of these techniques help, it's time to look outside of the Linux world. I've found that the Usenet FAQs provide an excellent source of information about free software. The comp.compilers FAQ has an extensive list of free language tools, including compilers and interpretors. Likewise, the rec.sport.football FAQ tells of an ftp site (ftp.vnet.com) that contains software for football fans and the comp.db FAQ has a large list of free database packages.

The hardest part about finding an FAQ is finding the name of a newsgroup it's been posted to. If your newsreader lets you search for newsgroups via a regular expression, that's the best way. Both rn and trn let you do this with the “a” command at the newsgroup selection level. Typing “a” followed by a regular expression returns the names of all newsgroups you have access to which satisfy that expression. For example, to look for newsgroups related to computer sound “a comp.*sound” (which returns 6 matches) would be a good way to start. If you're not familiar with regular expressions the GNU grep man page provides a good introduction, as did the introduction to grep in *Linux Journal* issue 18. If your newsreader doesn't let you search for newsgroups, try grepping the .newsrc file in your home directory (if you have one).

Once you get the name of the newsgroup whose FAQ you'd like to peruse, the rest is easy. The site rtfm.mit.edu collects all of the FAQs that are posted to the net and automatically updates them as new versions are released. This site is very busy and supplies a list of mirrors if you can't log in anonymously.

The directory /pub/usenet contains a directory for each newsgroup which has one or more FAQs. Those directories contain the FAQs themselves.

If you couldn't find the name of a newsgroup that might be helpful, all is not lost. Rtfm has a separate FAQ directory tree identical to Usenet's hierarchy. If you ftp in and cd to /pub/usenet-by-hierarchy you get a directory with a subdirectory for each top level news domain (such as comp, alt, rec and talk). Those subdirectories contain further subdirectories. For example, the newsgroup comp.sys.ibm.pc.soundcard.misc would have it's FAQ stored in the directory /pub/usenet-by-hierarchy/comp/sys/ibm/pc/soundcard/misc on rtfm.mit.edu.

Another resource for finding programs is the comp.archives newsgroup. It contains regular postings from a variety of archive sites detailing their contents. Reading through those posts or searching though it's archives (such as the one at ftp://wuarchive.wustl.edu/usenet/comp.archives) can help narrow down the sites to wade through.

The final choice (and the hardest to use) is the archie search service. While archie was designed for ftp searches it is an old service, and fairly hard to use. The various archie databases on the net each contain file lists from about 1200 ftp sites throughout the world, providing a total of around 2.5 million unique filenames that may be searched (the whole database is currently around 400MB). If a file can be ftped from the net, archie knows about it.

The World Wide Web provides the easiest way to conduct archie searches. A huge list of archie gateways on the web is available at web.nexor.co.uk/archie.html. While many of them support forms-based browsers, most also provide searchable indexes.

Several direct archie search clients are also available from archie.mcgill.ca in the /pub/archie-clients directory and come with some Linux distributions. They allow searching from either the command line or via a X search engine.

Finally, there are several telnet interfaces to archie databases. For example, telnetting to archie.mcgill.ca provides an interactive archie search session. While it can be somewhat cryptic, typing “help” should help you figure it out.

When you're searching archie it is important to remember that it just indexed file paths. If you know the file name you're looking for this isn't a problem, otherwise you have to guess what a likely name is.

You'll find that when you search on a term that finds matches, the matches will come in a flood. Using one of the archie WWW interfaces to search for “mixer” as a substring found 95 matches, which is what I had the match limit set to. All of the sites it found were in Europe, suggesting that archie hadn't begun to search any of the other continents yet!

The final thing to know about archie is that it can be very slow. It is an extremely popular way of searching the net due to the sheer volume of information it contains and the load on it results in search times of many minutes even at odd times of the night.

The Linux Software Map, the primary linux ftp sites, Usenet FAQs, comp.archives, and archie all provide useful ways of finding freely available Linux software on the net.

Erik Troan (ewt@redhat.com) maintains the sunsite.unc.edu /pub/Linux directory tree and is a developer at Red Hat Software.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

Advanced search

The Trouble With Live Data

David Bonn

Issue #24, April 1996

One use of such hacks is to break security.

live data, n. 1. Data that is written to be interpreted and takes over program flow when triggered by some un-obvious operation, such as viewing it. One use of such hacks is to break security. For example, some smart terminals have commands that allow one to download strings to program keys; this can be used to write live data that, when listed to the terminal, infects it with a security-breaking *virus* that is triggered the next time a hapless user strikes that key. For another, there are some well-known bugs in *vi* that allow certain texts to send arbitrary commands back to the machine when they are simply viewed. 2. In C code, data that includes pointers to function *hook*s (executable code). 3. An object, such as a *trampoline*, that is constructed on the fly by a program and intended to be executed as code. --From the Jargon file, version 3.2.0

Introduction (A Tale of Woe)

The link seemed innocent enough. All it said was:

```
Crusty the Clown fans should click here!!!
```

Perhaps I was a bit naive—or just stupid. But I clicked on the damned link. The actual transfer didn't take very long, and pageview (Sun's PostScript viewer) quickly brought up a picture of Crusty the Clown (from The Simpsons). The first inkling of any trouble was that my disk drive light came on. And stayed on. That almost never happens, especially without me doing something. I pulled up another xterm. Something was really funny; my tcsh configuration was all wrong...

My drive light went off. My home directory was gone.

Some fancy footwork with my sysadmin got my home directory back (fortunately it had been backed up the night before). Since the catastrophe happened early in the morning, nothing except the previous evening's e-mail was lost.

I was damned lucky.

All that had happened was that someone had embedded a command line in the PostScript image. The PostScript viewer I was using happily executed the command and 200 megabytes of my files were simply erased.

More Live Data

As the Internet becomes more and more sophisticated, rich and complex schemes of data interpretation become more common. Each of these schemes is a potential security hole. Conventional security mechanisms are less than effective at dealing with these problems.

The breadth of use of live data makes finding an effective solution quite hard. Users are demanding the newer, richer Internet services and site administrators cannot delay adding such services for very long. The widespread deployment of web browsers and MIME e-mail involves fairly substantial security risks which have many unintended consequences.

The (true) nightmare story introducing this article is a good example. Most PostScript viewers have a secure mode which disables the file writing and program-launching operations that are part of the PostScript language. However, my local configuration did not use the secure mode and I got burned by someone's idea of humor. Problems with embedded PostScript are well-known, however. Other tools which (at first) seem innocuous may also have potential problems.

The infamous Internet Worm of 1987 used a simple buffer-overrun bug in the finger daemon to compromise security on a great many computers. Historically, a great many C programs have had similar flaws—maybe not as dangerous to security as the finger bug, but often quite catastrophic in their own right.

Buffer-overrun Bugs sidebar

Consider the current state-of-the-art in web browsers. The vast majority of the web browsers currently on the market share a common ancestry with NCSA's Mosaic. Given the intense market pressures to bring web clients to market in today's Internet feeding frenzy and greased-pig-catching contest, it seems unlikely that extreme care has been taken in ensuring that web browsers

securely interpret HTML. Buffer overflow problems (which seem particularly likely in anchors) could give unscrupulous individuals a way to execute any arbitrary code sequence on target machines.

This problem is only getting worse. The Java language is currently exploding through the Internet world, and it seems likely that there will be a great many surprises associated with Java. Even though Java is intended to provide security mechanisms to prevent abuse, all it takes is one flaw to cause a major problem.

Another example of live data (which few people have considered) is the local variables list in Emacs. This is a highly convenient feature, as it allows per-file customization for the Emacs editor. For example, programmers can code their bracing and indentation preferences into the file, so other emacs users who edited the file are automatically able to follow those same bracing conventions. However, since many of the local "variables" are typically Emacs Lisp functions, there could be a substantial risk associated with merely viewing a file in Emacs—the same risks associated with viewing PostScript files with an unsafe viewer.

The Emacs problem is fairly manageable. If you set the variable **enable-local-variables** to **nil**, this feature of Emacs is disabled.

It seems that the biggest risks are associated with the most complex services. MIME e-mail provides a mechanism for launching viewer programs based on the contents of the e-mail. So, if you send a JPEG picture, a JPEG viewer will be launched. Thus, a very large (and rapidly expanding) set of programs can be involved—too many to ensure reasonable security. Some MIME configurations even allow the transmission and execution of shell scripts or Perl programs.

Some Non-Solutions

The cheapest, most reliable, and most secure components of any system are the ones that aren't there. Is it possible to solve the live data problem by avoiding the use of the advanced tools that pose the threat?

It probably isn't practical for a site administrator to outlaw such tools. For one thing, they are all too useful, and one has to weigh a possibly substantial security risk against a threat to an organization's competitiveness. In addition, it is unlikely that any but the most draconian site administrators could prevent users from acquiring their own personal web browsers or MIME e-mail clients. Fascist site administration might make matters worse, since the individuals who were using the outlawed tools would obviously not be informing site security staff of what they were doing.

Internet firewalls are becoming a popular security mechanism. However, it doesn't seem likely that they can protect against hostile live data. They could

completely block risky services (like e-mail and the Web), but if you completely block those services there isn't much point to being on the Internet. It also does not seem practical for a firewall to inspect all data coming from the net and looking for “dangerous” activities. For one thing, we don't have a good mechanism for distinguishing “friendly” live data from “unfriendly” live data—and the most dangerous live data doesn't look live at all. Another point to think about is that the effort involved in trying to solve this problem will likely put an unacceptable performance burden on the firewall.

Some Possible Solutions

There does not seem to be any “silver bullet” solution to this problem. However, there are some fairly simple steps that can be taken to provide reasonable protection:

- If possible, run your web browser and perform other possibly risky activities (e.g. viewing PostScript files, running programs you have downloaded) as another user ID that you use expressly for that purpose. This makes it less likely that major damage to your personal files or your system could occur.
- Save your “dot files” frequently. It may also help to allow only read-only access to such files. Dot files (such as your .profile, .emacs, .exrc, or .rhosts files) are frequent targets of live-data attacks.
- Probably the best advice is to be a little bit paranoid. If you get a large MIME attachment that appears to be a shell script, treat it the same way you would treat an armed bomb.

The best advice is not to avoid tools which use live data, but rather to use them very carefully. Being aware of the risks is probably the best defense. So have fun, and be careful out there.

David Bonn When he isn't busy skiing, he is usually fiddling around with Linux. When he isn't doing those two things, he is busy being president of Mazama Software Labs. Since David graduated from the University of Washington in 1986, most of his computer time has been spent working on networked systems.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Letters to the Editor

LJ Staff

Issue #24, April 1996

Readers sound off.

Arena

Regarding Phil Hughes' article in the December issue of *LJ*, please help push further development of Arena. Netscape is huge, slow, and our only option. I now use Lynx for almost all my browsing because Netscape is so slow on my system. Arena runs well, but cannot handle forms, one of the most functional features of a web browser. Like many people, I am not interested in background patterns and enhanced fonts. I want easy access to information, which I am unable to get with Netscape or the current Arena.

—Peter McArthur peterm@3rdplanet.com

More Ideas

Linux Journal has a sister publication called WEBsmith™, and its editor, Jon Gross, was recently involved in a World Wide Web conference. We asked him to respond:

I (and many others) agree with your summation of the status of the browser selection available at this point. In December, I was at the 4th International World Wide Web conference in Boston, and a group of us started talking, and decided it was time to start a Free Browser project, modeled on the Linux community, to address exactly this problem. We are putting a mildly cohesive framework together before "going public" but your input is certainly appreciated. The Linux community has much to offer this project, and I think we can definitely build a better mousetrap. A mailing list has been set up. See www.base.com/gordoni/web/www4-bof.html for more information.

—Websmith Magazine Editor, Jonathan Gross

Sticky Mistake

“The chmod Command” by Eric Goebelbecker (*LJ* #21) was an excellent introduction to the sometimes counterintuitive file and directory permissions of Unix-like operating systems. Near the end, however, there is a misstatement concerning the “sticky bit”. This bit actually has a very important, though obscure, use on Linux systems.

The sticky bit got its name on early Unix systems in the days before demand-paged virtual memory. If the sticky bit of an executable file was set, after the program exited a copy of its text segment was saved in the system's swap area, i.e., it would “stick around” for the next use. This feature was used to make frequently-run programs load a little bit faster. This meaning of the sticky bit is no longer particularly relevant.

Although it is not a part of the POSIX specification, recent System V and Berkeley Unix systems defined a new meaning for the sticky bit. If a directory's sticky bit is set, a file in the directory may be deleted only by the file's owner, by the directory's owner, or by root. This provides an additional measure of security for directories such as /tmp. Linux, of course, supports this useful feature. For example, on my system:

```
$ ls -ld /tmp
drwxrwxrwt  3 root   root   1024 Jan  1 09:49 /tmp
$ ls -l /tmp
-rw-------  1 roman  users     0 Jan  1 09:49 bar
-rw-------  1 root   root      0 Jan  1 09:47 foo
$ rm /tmp/bar
$ rm /tmp/foo
rm: /tmp/foo: Operation not permitted
```

Without the sticky bit, I would have been able to remove root's file from /tmp.

—Bill Roman roman@songdog.eskimo.com

Oops

Mea Culpa. *Thank you for pointing out that mistake. It should never have passed through editing.*

Things have changed

I read your article on LISP-Stat last August with much interest, and have just got around to trying it out. One inconsistency: the article claims that the “dld” library is essential. This is not true; if you build Lisp-Stat from scratch, it will use the now-standard “dlopen” method for dynamic loading (based on the ld.so library) if it is present. In fact, “dld” no longer seems to work with current “a.out” style static libraries, let alone ELF libraries.

I compiled the thing for ELF using gcc 2.7.0 and everything worked just great. The compile time is rather long (about an hour on a 486-33 with 20M memory), especially since a lot of the code is in LISP. All of the book examples worked perfectly. A marvellous way to review long-forgotten stats material.

You are doing an excellent job at presenting intriguing applications for Linux. Please keep up the great work!

—Rod Hallsworth rhallsw@synapse.net

Well, yes…

When the article was written, dld was required. At that time, ELF was still somewhat experimental and since we didn't know when it would be considered generally stable, we chose not to base articles on the assumption that it would become stable between the time the article was written and read. We have since started recommending that people switch to ELF when it is reasonably possible for them to do so, since it is now considered stable, and since the time will come (and obviously already is, to some extent) when applications are only available in ELF. Thanks for pointing out that it builds without a problem with ELF.

Picky Python

I found the Python examples largely worthless because of errors. Line 6 of the program listed on page 21 should read:

```
uid = `posix.getuid()`
```

Notice the (`) rather than (').

The largest problem was the format of the **StackingThings** program listed on page 23. Python is **VERY** sensitive to indentation. The attempt to format the program to fit the column “broke it real bad”.

Prior remarks aside, I found the Python article enjoyable and I am very pleased with your publication. Keep up the good work.

—Joe Kirby kirby@utk.edu

One and One

The “backticks” are really opening single quote characters in the courier font, which is suboptimal, but for which we are hard pressed to find a better replacement. If you look very, very carefully, you can see that the “backticks” are wider at the bottom than the top, and the “ticks” are wider at the top than

the bottom. We agree that you shouldn't have to look closely to see the difference, and we are working on a solution.

Our production team did indeed break the indentation in the **StackingThings** example, and we apologize. The production team at *LJ* usually does a very good job making things fit, but this time there were some significant slips...

The correct version of the StackingThings example can be retreived from our WWW server at www.ssc.com/lj/issue21/index.html.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Legal Battles Ended

Michael K. Johnson

Issue #24, April 1996

The three year criminal investigatin of the author of PGP - Pretty Good Privacy is over - the case was closed without prosecution.

The three-year criminal investigation of Philip Zimmermann, the author of Pretty Good Privacy (PGP), by the U.S. federal government was closed without prosecution. Mr. Zimmermann had been alleged to have violated the antiquated U.S. export control laws in releasing PGP.

A few days later, the U.S. Supreme court ruled in favor of Borland in the Lotus copyright infringement suit. This means that the court has again upheld the notion that a user interface is not copyright-able, which is certainly a relief to the free software community.

David 1, Goliath 0

Those who tuned into last month's **Stop the Presses** read about what I termed “The Desktop War”, and about two products which are attempting to bring MS Windows applications to Linux and other platforms: The freeware program Wine, and a commercial product called **TWIN XPDK**, created by a small company called Willows Software. There has been a recent flurry of activity at Willows.

Last month, the European Computer Manufacturers Association (ECMA) approved a new standard called APIW (Application Programming Interface for Windows) over the objections of Microsoft. This standard, authored by Willows, is now on track for ISO approval as an international standard.

David 2, Goliath 0

This month, Willows announced that they are releasing binaries and source code for XPDK (a toolkit for porting MS Windows applications to Linux and Unix

and soon Mac, OS/2, and Netware, which, not surprisingly, follows the APIW specification) free for non-commercial use. At the same time, they announced their extensive commercial support program, which is detailed on their World Wide Web site, www.willows.com

Willows has officially said that they do not support using XPDK for running off-the-shelf Windows program, because supporting it would be extremely time-intensive and take more resources than they have. However, they discovered during beta testing that many people joined the beta test group solely to attempt to run off-the-shelf binaries with the included “xwin” Windows program loader. By releasing the source code, as well as the binaries, free for non-commercial use, they allow technically capable users to support themselves and others, even using the xwin program to run off-the-shelf binaries. Essentially, the non-commercial users will be “paying” for the software by contributing bug reports, and bug fixes if they are capable.

While the xwin program currently only supports the 16-bit windows interface, the library can be compiled in a 16-bit or 32-bit version. Willows says that the full Win32s API will be supported by June, because they have already written most of the infrastructure that is necessary.

When asked whether he thinks Wine poses any threat to Willows' business, Rob Farnum, CEO of Willows, said that he doesn't think so, for a few reasons. Essentially, Wine has a much narrower scope: running windows binaries on a few Intel Unix platforms, and eventually providing a win32s library for most 32-bit Unix platforms. By contrast, XPDK can run Windows binaries on other processors, and is not X- or Unix-specific. It is already running on PowerPC Macs and is targeted to be released for OS/2, Netware, and both flavors of Macintosh this year. This doesn't mean that Willows doesn't take Wine seriously; they have paid close attention to its development and regularly benchmark XPDK against it.

When asked what pieces were currently missing from XPDK, Rob said that implementing OLE is the biggest technical challenge facing them. When asked whether he thought Microsoft's recent noises about OLE for Unix were meaningful, he said “No.” His reasons were:

- The projected release date is too late to be meaningful. The actual coding is being done by Software AG, who has licensed the OLE code from Microsoft, and who is projecting a late 1997 or 1998 release date.
- OLE for Unix is *already* available from Mainsoft and Bristol, as part of their Windows API on Unix packages.
- OLE is so tied to Windows internals that it isn't clear what OLE on Unix would mean, anyway, and Microsoft has not made this clear.

He suggests that the announcement was a tactical move by Microsoft against CORBA (Common Object Request Broker Architecture) and the OMG (the Object Management Group with supervises CORBA). Microsoft would like to be able to ignore its promises to provide an ORB (Object Request Broker) for Windows.

The Latest, The Greatest

Linus Torvalds has stated that another code freeze is about to go into effect, which will result in a new stable version of Linux. Linus has stated that he is leaning towards calling this new version **2.0** instead of **1.4** (just call it **Linux 96**...), since it now supports multiple platforms and multi-processing.

A few months after the new stable release is made, *Linux Journal* will include an article on how to upgrade safely from version 1.2, just as we did for upgrading from 1.0 to 1.2. We will also detail the changes. Here's a preview:

Improvements in the new version will include (but not be limited to!) support for more hardware, bug fixes, major performance enhancements (some benchmarks have improved by 200-300%), and new networking features. In particular, new filesystem support has been added to allow Linux systems to mount Novell file servers (**ncpfs**) and Windows shared volumes (**smbfs**).

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Dynamic Kernels: Discovery

Alessandro Rubini

Issue #24, April 1996

This article, the second of five, introduces part of the actual code to create custom module implementing a character device driver. It describes the code for module initialization and cleanup, as well as the open and close system calls.

The last issues *Kernel Korner* introduced problems related to loading and unloading a custom module, but didn't uncover the code to actually perform these tasks. This time, we are going to look at some finer details of module-writing, in order to begin showing the actual code for our character device driver.

Load-time Configuration

Although a smart driver should be able to autodetect the hardware it looks for, autodetection is not always the most sensible implementation, because it may be tricky to design. It is wise to provide a way to specify as many details as possible at load time, in order to test your driver before scratching your head and deciding to implement autodetection. Moreover, autodetection may fail if “similar” hardware is installed in the computer. A minor project can simply avoid autodetection altogether.

To configure the driver at load time, we'll exploit insmod's capability to assign integer values to arbitrary variables in the driver. We'll thus declare a (public) integer variable for each configurable item, and we'll make sure that the default value will trigger autodetection.

Configuring multiple boards at load time is left as an exercise to the reader (after reading the manpage for **insmod**); this implementation allows specification of a single board: for the sake of simplicity additional boards are only reachable through autodetection.

Choosing Names

The kernel is a complex application, and it is vital to keep its namespace as tidy as possible. This means both to use private symbols wherever it is possible, and to use a common prefix for all the symbols you define.

A production environment will only declare **init_module()** and **cleanup_module()**, which are used to load and unload the driver, and any load-time configuration variables. Nothing else needs to be public, because the module is accessed through pointers, not by name.

However, when you are developing and testing your code, you'll need your functions and data structures in the public symbol table in order to access them with your favorite debugging tool.

The easiest way to accomplish this dual need is to always use your own prefix in names, declare all of your symbols **Static** (note the capital `S'), and include the following five lines at the top of your driver:

```
#ifdef DEBUG_modulename
# define Static /* nothing */
#else
# define Static static
#endif
```

Real static symbols (such as persistent local variables) may thus be declared **static**, while debuggable symbols are declared **Static**

The init_module() Function

In this page, the whole code for the initialization function is uncovered. This is skeletal code, as the **skel** name suggests: a real-world device usually has slightly more than two I/O ports.

The most important thing to remember here is to release all the resources you already asked for whenever you find an error condition. This kind of task is well handled by the (otherwise unloved) **goto** statement: code duplication is avoided by jumping to the resource-release part of the function in case of error.

The fragment of code shown accepts load-time configuration for the major number, for the base address of the board's I/O ports, and for the IRQ number. For each "possible" board (in the I/O space), the autodetection function is called. If no boards are detected, **init_module()** returns **-ENODEV** to tell insmod that no devices are there.

Sometimes it is wise to allow the driver to be loaded even if its hardware is not installed in the computer. I implement such code in order to develop most of

my driver at home. The trick is to have a configuration variable (**skel_fake**) which allows you to fake a nonexistent board. You can look at the implementation in my own drivers. “Faking boards” is a powerful way to start writing code before you get the hardware, or to test support for two boards even if you only own one of them.

The role of **cleanup_module()** is to shut down the device and release any resources allocated by **init_module()**. Our sample code cycles through the array of boards and releases I/O ports and the IRQ, if any. Finally, the major number is released. The initial check for **MOD_IN_USE** is redundant if you're running a recent kernel, but a wise thing to put in production code, because your customers or users may be running old Linux kernels.

The sample code for **init_module()** and **cleanup_module()** is shown in Listing 1. The prefix **skel_** is used for all non-local names. The code here is quite simplified, in that it lacks some error-checking, which is vital in production-quality source code.

Autodetecting the Device

init_module() calls the function **skel_find()** to perform the dirty task of detecting if a board is there. The function is very device specific, because each device must be probed for its peculiar features; thus, I won't try to show code to perform the actual probing, but only IRQ autodetection.

Unfortunately, some peripherals can't tell which IRQ line they're configured to use, thus forcing the user to write the IRQ number on the command line of insmod, or to hardcode the number in the software itself. Both these approaches are bad practice, because you just can't plug the board (after setting the jumpers) and load the driver. The only way to autodetect the IRQ line for these devices is a trial-and-error technique, which is, of course, only viable if the hardware can be instructed to generate interrupts.

The code in Listing 2 shows **skel_find()**, complete with IRQ autodetection. Some details of IRQ handling may appear obscure to some readers, but they will be clarified in the next article. To summarize, this code cycles through each of the possible IRQ lines, asking to install a handler, and looks to see if interrupts are actually generated by the board.

The field **hwirq** in the hardware structure represents the *useable* interrupt line, while the field **irq** is only valid when the line is active (after **request_irq()**). As explained in the last issue, it makes no sense to keep hold of an IRQ line when the device is not in use; that's why two fields are used.

Please note that I wrote this code as a work-around for the limitations of one of my hardware boards; if your hardware is able to report the IRQ line it's going to use, it's much better to use *that* information instead. The code is quite stable, anyway, if you are able to tailor it to your actual hardware. Fortunately, most good hardware is able to report its own configuration.

fops and filp

After the module has been loaded and the hardware has been detected, we must see *how* the device is acted upon. This means introducing the role of **fops** and **filp**: these little beasts are the most important data structures—actually, variable names—used in interfacing the device driver with the kernel.

fops is the name usually devoted to a **struct file_operations**. The structure is a **jump table** (structure of pointers to functions), and each field refers to one of the different operations performed on a filesystem node (**open()**, **read()**, **ioctl()**, etc.).

A pointer to your own **fops** is passed to the kernel by means of **register_chrdev()**, so that your functions will be called whenever one of your nodes is acted upon. We already wrote that line of code, but didn't show the actual **fops**. Here it is:

```
struct file_operations skel_fops {
  skel_lseek,
  skel_read,
  skel_write,
  NULL,       /* skel_readdir */
  skel_select,
  skel_ioctl,
  skel_mmap,
  skel_open,
  skel_close
};
```

Each **NULL** entry in your fops means that you're not going to offer that functionality for your device (**select** is special, in this respect, but I won't expand on it), each non-**NULL** entry must be a pointer to a function implementing the operation for your device. Actually, there exist a few more fields in the structure, but our example will live with the default **NULL** value (the C compiler fills up an incomplete structure with zero bytes without issuing any warning). If you are really interested in them, you can look at the structure's definition in **<linux/fs.h>**. **filp** is the name usually devoted to one of the arguments passed by the kernel to any function in your fops, namely a **struct file ***. The **file** structure is used to keep all the available status information about an “open file”, beginning with a call to **open()** and up to a call to **close()**. If the device is opened multiple times, different **filp**s will be used for each instance: this means that you'll need to use your own data structure to keep hardware information about your devices. The code fragments within this installment already use an

array of **Skel_Hw**, to hold information about several boards installed on the same computer. What is missing, then, is a way to embed hardware information in the **file** structure, in order to instruct the driver to operate on the right device. The field **private_data** exists in **struct file** just for that task, and is a pointer to **void**. You'll make **private_data** point to your hardware information structure when **skel_open()** gets invoked. If you need to keep some extra information private to each filp (for example, if two device nodes access the same hardware in two different ways), then you'll need a specific structure for **private_data**, which must be **kmalloc()**ed on open and **kfree()**ed on close. The implementations of open() and close() that we'll see later, work in this way.

Using Minor Numbers

In the last article I introduced the idea of minor device numbers, and it is now high time to expand on the topic.

If your driver manages multiple devices, or a single device but in different ways, you'll create several nodes in the **/dev** directory, each with a different minor number. When your open function gets invoked, then, you can examine the minor number of the node being opened, and take appropriate actions.

The prototypes of your open and close functions are

```
int skel_open (struct inode *inode,
               struct file *filp);
void skel_close (struct inode *inode,
                 struct file *filp);
```

and the minor number (an unsigned value, currently 8 bits) is available as **MINOR(inode->i_rdev)**. The **MINOR** macro and the relevant structures are defined within **<linux/fs.h>**, which in turn is included in **<linux/sched.h>**.

Our skel code (Listing 3) will split the minor number in order to manage both multiple boards (using four bits of the minor), and multiple modes (using the remaining four bits). To keep things simple we'll only write code for two boards and two modes. The following macros are used:

```
#define SKEL_BOARD(dev) (MINOR(dev)&0x0F)
#define SKEL_MODE(dev)  ((MINOR(dev)>>4)&0x0F)
```

The nodes will be created with the following commands (within the **skel_load** script, see last month's article):

```
mknod skel0    c $major  0
mknod skel0raw c $major  1
mknod skel1    c $major 16
mknod skel1raw c $major 17
```

But let's turn back to the code. This **skel_open()** sorts out the minor number and folds any relevant information inside the filp, in order to avoid further overhead when read() or write() will be invoked. This goal is achieved by using a **Skel_Clientdata** structure embedding any filp-specific information, and by changing the pointer to your **fops** within the **filp**; namely, **filp->f_op**.

Changing values within **filp** may appear a bad practice, and it often is; it is, however, a smart idea when the file operations are concerned. The **f_op** field points to a static object anyways, so you can modify it lightheartedly, as long as it points to a valid structure; any subsequent operation on the file will be dispatched using the new jump table, thus avoiding a lot of conditionals. This technique is used within the kernel proper to implement the different memory-oriented devices using a single major device number.

The complete skeletal code for **open()** and **close()** is shown in Listing 3; the **flags** field in the clientdata will be used when **ioctl()** is introduced.

Note that the **close()** function shown here should be referred to by both **fops**s. If different **close()** implementations are needed, this code must be duplicated.

Multiple- or Single-open?

A device driver should be a policy-free program, because policy choices are best suited to the application. Actually, the habit of separating policy and mechanism is one of the strong points of Unix. Unfortunately, the implementation of **skel_open()** leads itself to policy issues: is it correct to allow multiple concurrent opens? If yes, how can I handle concurrent access in the driver?

Both single-open and multiple-open have sound advantages. The code shown for **skel_open()** implements a third solution, somewhat in-between.

If you choose to implement a single-open device, you'll greatly simplify your code. There's no need for dynamic structures because a static one will suffice; thus, there's no risk to have memory leakage because of your driver. In addition, you can simplify your **select()** and data-gathering implementation because you're always sure that a single process is collecting your data. A single-open device uses a boolean variable to know if it is busy, and returns **-EBUSY** when open is called the second time. You can see this simplified code in the busmouse drivers and lp driver within the kernel proper.

A multiple-open device, on the other hand, is slightly more difficult to implement, but much more powerful to use for the application writer. For example, debugging your applications is simplified by the possibility of keeping a monitor constantly running on the device, without the need to fold it in the

application proper. Similarly, you can modify the behaviour of your device while the application is running, and use several simple scripts as your development tools, instead of a complex catch-all program. Since distributed computation is common nowadays, if you allow your device to be opened several times, you are ready to support a cluster of cooperating processes using your device as an input or output peripheral.

The disadvantages of using a conventional multiple-open implementation are mainly in the increased complexity of the code. In addition to the need for dynamic structures (like the **private_data** already shown), you'll face the tricky points of a true stream-like implementation, together with buffer management and blocking and non-blocking read and write; but those topics will be delayed until next month's column.

At the user level, a disadvantage of multiple-open is the possibility of interference between two non-cooperating processes: this is similar to cat-ing a tty from another tty—input may be delivered to the shell or to cat, and you can't tell in advance. [For a demonstration of this, try this: start two xterms or log into two virtual consoles. On one (A), run the **tty** command, which tells you which tty is in use. On the other (B), type **cat /dev/*tty_of_A***. Now go to A and type normally. Depending on several things, including which shell you use, it may work fine. However, if you run **vi**, you will probably see what you type coming out on B, and you will have to type **^C** on B to be able to recover your session on A—ED]

A multiple-open device can be accessed by several different users, but often you won't want to allow different users to access the device concurrently. A solution to this problem is to look at the **uid** of the first process opening the device, and allow further opens only to the same user or to root. This is not implemented in the skel code, but it's as simple as checking **current->euid**, and returning **-EBUSY** in case of mismatch. As you see, this policy is similar to the one used for ttys: login changes the owner of ttys to the user that has just logged in.

The **skel** implementation shown here is a multiple-open one, with a minor addition: it assures that the device is “brand new” when it is first opened, and it shuts the device down when it is last closed.

This implementation is particularly useful for those devices which are accessed quite rarely: if the frame grabber is used once a day, I don't want to inherit strange setting from the last time it was used. Similarly, I don't want to wear it out by continuously grabbing frames that nobody is going to use. On the other hand, startup and shutdown are lengthy tasks, especially if the IRQ has to be detected, so you might not choose this policy for your own driver. The field

usecount within the hardware structure is used to turn on the device at the first open, and to turn it off on the last close. The same policy is devoted to the IRQ line: when the device is not being used, the interrupt is available to other devices (if they share this friendly behaviour).

The disadvantages of this implementation are the overhead of the power cycles on the device (which may be lengthy) and the inability to configure the device with one program in order to use it with another program. If you need a persistent state in the device, or want to avoid the power cycles, you can simply keep the device open by means of a command as silly as this:

```
sleep 1000000 < /dev/skel0 &
```

As it should be clear from the above discussion, each possible implementation of the **open()** and **close()** semantics has its own peculiarities, and the choice of the optimum one depends on your particular device and the main use it is devoted to. Development time may be considered as well, unless the project is a major one. The **skel** implementation here may not be the best for your driver: it is only meant as a sample case, one amongst several different possibilities.
Additional Information

Alessandro Rubini (rubini@ipvvis.unipv.it) Programmer by chance and Linuxer by choice, Alessandro is taking his PhD course in computer science and is breeding two small Linux boxes at home. Wild by his very nature, he loves trekking, canoeing, and riding his bike.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Prime Time Freeware for UNIX

Preston Brown

Issue #24, April 1996

The back cover claims that the collection of software is geared towards programmers and users alike. But looks aren't everything; it's what is inside that counts with a package like this.

Editor: Richard Morin

ISBN: 1-881957-18-7

Price: $50

Reviewer: Preston Brown

As I unwrapped the package which the post office had delivered, I was preparing myself for just another carbon copy of the sunsite.unc.edu archives, or perhaps some other big Unix software site. However, as I removed the book from the shipping material and observed it for the first time, I had a strong suspicion I was in for something much more professional, and thus useful, in nature.

Prime Time Freeware (PTF) has now changed the format of the book that acompanies that CDROM to the familiar glossy paperback cover and standard size that most computer books seem to come in today, so it should fit in comfortably on your shelf. The back cover claims that the collection of software is geared towards programmers and users alike. But looks aren't everything; it's what is inside that counts with a package like this.

Included Materials

Prime Time Freeware for Unix includes not only the book, but of course a plethora of UNIX software, comfortably housed on two ISO-9660 CD-ROMs. The “version” of the package that I reviewed was 4-2, but since it is updated every

few months, a subscription buying plan is available. New issues will arrive automatically, and a one-month trial period with a money-back guarantee is standard. By the time you read this, the next issue should be out.

The CDs included are filled to the brim; PTF makes full use of the medium and doesn't skimp. Uncompressed, there is about about 5 gigabytes of information. Because the package is targeted at most Unix platforms, and is not limited to Linux, there are no binary packages; everything is distributed in source code form. This fact alone may turn off some potential buyers, but learning to compile code is an essential basic of Linux education, and should not really act as a deterrent.

The book, in addition to acting as good installation and troubleshooting guide, contains descriptions of all the packages on the disk, as well as their size and location. An index is included for easy reference. Finally, a good deal of space is devoted to describing the nature of free software itself, and it is clear that PTF is very devoted to the cause of the Free Software Foundation and independent programmers the world over.

Use and Installation of Software

Unfortunately, I encountered a problem early in my evaluation of the software, but one that was easily fixed. A slight flaw in the mastering process of some of the version 4-2 discs made it necessary to install a patch to the Linux kernel so that the discs could be successfully mounted. Needless to say, this was a pain, but the patch was included and was not difficult to install. PTF assured me that future issues would not have this problem.

Another drawback, while not an inherent problem, can still be slightly annoying. The PTF discs stick to the strict ISO-9660 format. This means that filenames must comply with the standard MS-DOS 11 character filenames, with no “funny” characters. PTF's decision to not use the “Rock Ridge Extensions,” which so many of us are used to, is a result of a lack of support for the extensions across all the platforms the package is targeted at.

Once the disc was mounted, I had to run a simple shell script which set up certain environment variables and the like to “customize” the disc for the operating system (in this case, of course, Linux). Navigating the discs to find software I wanted was fairly easy. Several methods are provided, including a detailed description database with paths, a more simplified database, and a HTML hypertext version of the database. A keyword index makes searching the detailed descriptions fairly easy. Information on using these databases is all well described in the accompanying book.

Just about any kind of software you are interested in is included on the discs, and it is all *very* up-to-date as well. For the user, there are databases (including Postgres and Onyx), archiving and compression tools, simple spreadsheets, editors and formatters (all flavors of Emacs, (La)TeX, troff), and graphics tools (data plotting/drawing, image manipulation, modelers and renderers). For the system administrator, there is plenty of communications stuff, including FAX tools, everything you could ever need for email, Usenet News, and the Athena networking suite.

Programmers will be delighted by the vast array of libraries (including graphics, GUI, etc.), and there are over 100 compilers and interpreters to choose from—some familiar, some relatively obscure. Few people have heard of CLU, but the latest version of gcc can be had here as well. There are also plenty of debuggers and profilers, syntax checkers, and the like. The full source code for X11R6 is here too. Math tools include the popular Scilab and Pari as well as many others, and for the scientist there is stuff for astronomy, chemistry, and even geology. Last, several operating systems (more or less complete) are included, like Andrew from Carnegie Mellon University, the OSF version of the Mach kernel, BSD 4.4lite, and Condor. Chances are, if you want it, it is here.

Installation of the packages is fairly primitive because of the source code format. Packages can be copied to hard drive, unarchived, and compiled by hand, or a simple included utility can be used to copy and unpack them. Either way, compiling and using them is up to you. However, these packages have all been tested and evaluated by the people at PTF, and would not be on the disc if they did not work.

Summary

Different parts of this disc will make different people happy. But the key point is that there is something (no, *plenty* of things) for everybody. The fact that all the packages are well documented and up-to-date is an additional big plus. Instead of yet another dump of some FTP site, we have two logically organized and planned discs which makes finding what you want more intuitive and easy than an Archie search.

I would tend to recommend the discs more for the programmer or hacker than end-user, despite the number of “user programs” included. For anyone who does any programming or scientific work, the discs can be quite helpful. I admit to using several of the packages myself off the discs. Also, anyone who doesn't have a fast Internet connection will appreciate how quickly they can have access to all the latest Unix software with the PTF package. It takes the drudgery out of FTPing and downloading. In short, while the discs are not a revolutionary breakthrough, they are definitely a big step for UNIX software

packaging and free software in general. If you think you might use it, get it. You won't be disappointed. However, if you neither need nor want the convenience of good package descriptions and organization, stick to your favorite FTP sites and save your money.

Preston Brown (preston.brown@yale.edu) is a sophomore computer science student at Yale University. He first discovered Linux with an early TAMU release in late 1992, and has been using it ever since.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

New Products

LJ Staff

Issue #24, April 1996

MerTech Credit Card Transaction Server, Linux Installation and Beyond and more.

MerTech Credit Card Transaction Server

MerTech Software & Consulting has announced the MerTech Credit Card Transaction Server for Unix and Linux based systems. The MerTech Credit Card Transaction Server combines with the Visa POS-port device to provide high speed, low cost, online credit card authorization and processing. Credit card transactions can be sent to the server over a local area network, using TCP/IP sockets, or in a batch file. Typical response times are comparable to those provided with an expensive leased-line service. The server can be easily interfaced with most database front-ends, and CGI scripts (for use with a Secure Web Server). Client APIs are available on most platforms.

Contact: MerTech Software & Consulting, 456 North 50 East, Orem, UT 84057. Phone: 801-368-9212. E-mail: mertech@xmission.com . URL: www.xmission.com/~mertech/.

Linux Installation and Beyond Video Available

Yggdrasil Computing has begun shipping its video seminar on installing Linux. *Linux Installation and Beyond* is a two and a half hour instructional video available on VHS in the NTSC format (the format used in the Americas and many industrialized Asian countries). A PAL version will be available soon.

Installing Linux for the first time can be intimidating, even for an experienced user. Seeing the installation of Linux first can save you time and help you better configure your system. If you already have Linux installed, you may find it informative to see the way someone else has configured their system or to quickly see the installation of a different Linux distribution without disrupting your existing system. Price: $24.95.

Contact: Yggdrasil Computing, 4880 Stevens Creek Blvd., Suite 205, San Jose, CA 95129-1024. Phone: 1-800-261-6630. E-mail: info@yggdrasil.com . URL: www.yggdrasil.com.

THOUGHT Inc. Releases Nutmeg

THOUGHT Inc. has announced the release of Nutmeg, an extensible Collection Class library for list management in Java. This release of Nutmeg works with the Java Beta 2 API, and applications written with Nutmeg require a runtime environment such as Netscape 2.0 or the Java Developer's Kit. These two runtimes are available on various platforms, including Windows NT, Windows 95, Solaris, and Linux. The Nutmeg Library contains classes such as Bag, Dictionary, IndexedCollection, OrderedCollection, Set, and many others. Nutmeg stays true to the Smalltalk Class Structure and API while utilizing features of Java such as multi-threading and exceptions. Price: Student and Non-commercial developers, binary license only—$49.00. Commercial developers and corporations, binary only—$495.00, source and binary—$995.00.

Contact: THOUGHT Inc., 2222 Leavenworth St., Suite 304, San Francisco, CA 94133. E-mail: nutmeg@thoughtinc.com . URL: www.thoughtinc.com/~nutmeg/.

Across Lite for Linux Released

Literate Software Systems has announced the release of version 1.0 of Across Lite for Linux. Across Lite is a multi-platform program for solving crossword puzzles online. Supported platforms include Linux, MS Windows, Mac and Solaris. Across Lite is part of LitSoft's Across product line. A complimentary copy of Across Lite is available for personal use from LitSoft's web site.

Contact: Literate Software Systems, P.O. Box 40, Morris Plains, NJ 07950-0040. URL: www.litsoft.com/.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Consultants Directory

> This is a collection of all the consultant listings printed in *LJ* 1996. For listings which changed during that period, we used the version most recently printed. The contact information is left as it was printed, and may be out of date.

ACAY Network Computing Pty Ltd
Australian-based consulting firm specializing in: Turnkey Internet solutions, firewall configuration and administration, Internet connectivity, installation and support for CISCO routers and Linux.

Address:
Suite 4/77 Albert Avenue, Chatswood, NSW, 2067, Australia
+61-2-411-7340, FAX: +61-2-411-7325
sales@acay.com.au
http://www.acay.com.au

Aegis Information Systems, Inc.
Specializing in: System Integration, Installation, Administration, Programming, and Networking on multiple Operating System platforms.

Address:
PO Box 730, Hicksville, New York 11802-0730
800-AEGIS-00, FAX: 800-AIS-1216
info@aegisinfosys.com
http://www.aegisinfosys.com/

American Group Workflow Automation
Certified Microsoft Professional, LanServer, Netware and UnixWare Engineer on staff. Caldera Business Partner, firewalls, pre-configured systems, world-wide travel and/or consulting. MS-Windows with Linux.

Address:
West Coast: PO Box 77551, Seattle, WA 98177-0551
206-363-0459
East Coast: 3422 Old Capitol Trail, Suite 1068, Wilmington, DE 19808-6192
302-996-3204
amergrp@amer-grp.com
http://www.amer-grp.com

Bitbybit Information Systems
Development, consulting, installation, scheduling systems, database interoperability.

Address:
Radex Complex, Kluyverweg 2A, 2629 HT Delft, The Netherlands
+31-(0)-15-2682569, FAX: +31-(0)-15-2682530
info@bitbybit-is.nl

Celestial Systems Design
General Unix consulting, Internet connectivity, Linux, and Caldera Network Desktop sales, installation and support.

Address:
60 Pine Ave W #407, Montréal, Quebec, Canada H2W 1R2
514-282-1218, FAX 514-282-1218
cdsi@consultan.com

CIBER*NET
General Unix/Linux consulting, network connectivity, support, porting and web development.

Address:
Derqui 47, 5501 Godoy Cruz, Mendoza, Argentina
22-2492
afernand@planet.losandes.com.ar

Cosmos Engineering
Linux consulting, installation and system administration. Internet connectivity and WWW programming. Netware and Windows NT integration.

Address:
213-930-2540, FAX: 213-930-1393
76244.2406@compuserv.com

Ian T. Zimmerman
Linux consulting.

Address:
PO Box 13445, Berkeley, CA 94712
510-528-0800-x19
itz@rahul.net

InfoMagic, Inc.
Technical Support; Installation & Setup; Network Configuration; Remote System Administration; Internet Connectivity.

Address:
PO Box 30370, Flagstaff, AZ 86003-0370

602-526-9852, FAX: 602-526-9573
support@infomagic.com

Insync Design
Software engineering in C/C++, project management, scientific programming, virtual teamwork.

Address:
10131 S East Torch Lake Dr, Alden MI 49612
616-331-6688, FAX: 616-331-6608
insync@ix.netcom.com

Internet Systems and Services, Inc.
Linux/Unix large system integration & design, TCP/IP network management, global routing & Internet information services.

Address:
Washington, DC-NY area,
703-222-4243
bass@silkroad.com
http://www.silkroad.com/

Kimbrell Consulting
Product/Project Manager specializing in Unix/Linux/SunOS/Solaris/AIX/HPUX installation, management, porting/software development including: graphics adaptor device drivers, web server configuration, web page development.

Address:
321 Regatta Ct, Austin, TX 78734
kimbrell@bga.com

Linux Consulting / Lu & Lu
Linux installation, administration, programming, and networking with IBM RS/6000, HP-UX, SunOS, and Linux.

Address:
Houston, TX and Baltimore, MD
713-466-3696, FAX: 713-466-3654
fanlu@informix.com
plu@condor.cs.jhu.edu

Linux Consulting / Scott Barker
Linux installation, system administration, network administration, internet connectivity and technical support.

Address:
Calgary, AB, Canada
403-285-0696, 403-285-1399
sbarker@galileo.cuug.ab.ca

LOD Communications, Inc
Linux, SunOS, Solaris technical support/troubleshooting. System installation, configuration. Internet consulting: installation, configuration for networking hardware/software. WWW server, virtual domain configuration. Unix Security consulting.

Address:
1095 Ocala Road, Tallahassee, FL 32304
800-446-7420
support@lod.com
http://www.lod.com/

Media Consultores
Linux Intranet and Internet solutions, including Web page design and database integration.

Address:
Rua Jose Regio 176-Mindelo, 4480 Cila do Conde, Portugal
351-52-671-591, FAX: 351-52-672-431
http://www.clubenet.com/media/index.html/

Perlin & Associates
General Unix consulting, Internet connectivity, Linux installation, support, porting.

Address:
1902 N 44th St, Seattle, WA 98103
206-634-0186
davep@nanosoft.com

R.J. Matter & Associates
Barcode printing solutions for Linux/UNIX. Royalty-free C source code and binaries for Epson and HP Series II compatible printers.

Address:
PO Box 9042, Highland, IN 46322-9042
219-845-5247
71021.2654@compuserve.com

RTX Services/William Wallace
Tcl/Tk GUI development, real-time, C/C++ software development.

Address:
101 Longmeadow Dr, Coppell, TX 75109
214-462-7237
rtxserv@metronet.com
http://www.metronet.com/~rtserv/

Spano Net Solutions
Network solutions including configuration, WWW, security, remote

system administration, upkeep, planning and general Unix consulting. Reasonable rates, high quality customer service. Free estimates.

Address:
846 E Walnut #268, Grapevine, TX 76051
817-421-4649
jeff@dfw.net

Systems Enhancements Consulting
Free technical support on most Operating Systems; Linux installation; system administration, network administration, remote system administration, internet connectivity, web server configuration and integration solutions.

Address:
PO Box 298, 3128 Walton Blvd, Rochester Hills, MI 48309
810-373-7518, FAX: 818-617-9818
mlhendri@oakland.edu

tummy.com, ltd.
Linux consulting and software development.

Address:
Suite 807, 300 South 16th Street, Omaha NE 68102
402-344-4426, FAX: 402-341-7119
xvscan@tummy.com
http://www.tummy.com/

VirtuMall, Inc.
Full-service interactive and WWW Programming, Consulting, and Development firm. Develops high-end CGI Scripting, Graphic Design, and Interactive features for WWW sites of all needs.

Address:
930 Massachusetts Ave, Cambridge, MA 02139
800-862-5596, 617-497-8006, FAX: 617-492-0486
comments@virtumall.com

William F. Rousseau
Unix/Linux and TCP/IP network consulting, C/C++ programming, web pages, and CGI scripts.

Address:
San Francisco Bay Area
510-455-8008, FAX: 510-455-8008
rousseau@aimnet.com

Zei Software
Experienced senior project managers. Linux/Unix/Critical business software development; C, C++, Motif, Sybase, Internet connectivity.

Address:
2713 Route 23, Newfoundland, NJ 07435
201-208-8800, FAX: 201-208-1888
art@zei.com

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.