LINUX JOURNAL

Advanced search

Linux Journal Issue #37/May 1997

Features

Linux On the PS/2 *by David Weis*
: While still a challenge, it has recently become much easier to install Linux on a PS/2 with an ESDI drive. Here's how.

Linux/m68k: Linux on the Motorola 68000 Processor *by Chris Lawrence*
: In the midst of all the attention given to ports to evermore exotic hardware, it's easy to overlook the first production quality port: Linux/m68k. The current version is the most stable yet.

Native Linux on the PowerPC *by Cort Dougan*
: Users of the PowerPC no longer have to settle for less—here's how to run Linux on machines with the PCI bus.

Linux? On the Macintosh? with Mach? *by Vicki Brown*
: The answer is an emphatic yes: Disover MkLinux.

News & Articles

Tcl/Tk with C for Image Processing *by Siome Klein Goldenstein*
Internet Servers in Perl *by Mike Mull*
An Interview with DEC *by John “maddog” Hall and David Rusling*
Safely Running Programs as root *by Phil Hughes*
LJ Interviews Przemek Klosowski *by Marjorie Richardson & Lydia Kinata*
Python Update *by Andrew Kuchling*

Linux on the PS/2

David Weis

Issue #37, May 1997

While running Linux on a PS/2 is not the usual choice, it is getting easier.

The PS/2 line historically has not been able to run Linux due to the Microchannel (MCA) Bus used in it. Technical specifications were difficult to get from IBM. There has been support available to run Linux on MCA machines since about 1994, but it was difficult to install and required much patching by the user. That situation has changed, and MCA machines are relatively easy to install. Make no mistake, though, if you plan to run Linux on your PS/2, you are still in for a challenge.

Supported hardware for MCA machines includes 3Com and SMC-WD Ethernet cards, IBM token ring cards, many SCSI cards and various other cards. Most lower-end PS/2s have ESDI drive controllers, which are not supported in the standard Linux kernel. Also, bus differences require the kernel to do other timing-related functions not necessary on either an ISA or a PCI bus.

This article gives fairly detailed instructions on installing Slackware on a PS/2 with an ESDI drive. It was tested on my 55SX with 4MB of RAM and a 60MB hard drive. The Slackware CD was NFS mounted from another machine through an SMC Ethernet card.

Note, this would probably not be a good first installation for anyone. It assumes familiarity with Slackware, and some steps normally performed by the the setup program must be done manually.

First, export the Slackware directories.

Before starting, as with any normal Slackware installation, you will need a boot disk, a root disk and one more disk with device files, a modified fdisk and a modified LILO. These files are available at ftp://glycerine.cetmm.uni.edu/pub/slackware/.

Begin by downloading ps2-boot.gz, color.gz and esdi_slack.tgz. ps2-boot and color must be decompressed; otherwise, you will probably get some errors on the boot. Next:

```
cat ps2-boot >/dev/fd0
```

Then, switch disks:

```
cat color>/dev/fd0
```

Then, switch disks:

```
cat esdi_slack.tgz >/dev/fd0
```

Be sure to label these disks.

Drive geometry detection does not always work, so you'll need to know your cylinders, heads and sectors. Below is a table for IBM drives.

```
size    command line
30 MB
60 MB   ed=58,64,32
```

Insert the Boot disk and turn on your machine. When you get to the LILO prompt enter **ramdisk ed=58** (or 64 or 32—choose appropriate geometry). Watch the kernel messages go by to see if your hardware was properly detected. Your machine may seem to hang while resetting the ESDI drive, but it can take up to 15 seconds. Eventually, you will get a login prompt. Log in as root. Run **fdisk /dev/eda** (the first ESDI device). Delete all the partitions on the drive. Now you will need to make a root partition and a swap partition. For the 60MB drive, I recommend 50MB for the root and 8 for the swap partition. To set up the partitions, pick: new partition, primary partition, 1st primary partition, start at cylinder 1, end at cylinder 50. This is your root. Now pick: new again, primary partition, 2nd primary partition, start at cylinder 51, and end at cylinder 58. Also select “change the type of partition 2 to 82 (Linux Swap)”. Print the partition table to make sure there are no obvious problems. Check how many blocks are in the swap partition, because you will need that information later. With the numbers above it should be 8192. Go ahead and write it to disk.

If your machine is like mine, it doesn't have a lot of RAM installed. In order to run the setup program you will need to activate the swap partition. To do that, run **mkswap /dev/eda2** (number of blocks). After some disk activity, run **swapon /dev/eda2**. Now your machine is ready for the setup program.

Run setup, the Slackware install program. Choose to add a swap partition. It will find the /dev/eda2 partition itself. Be sure to pick “no” when asked to run mkswap or swapon. Running these twice will cause problems. Now select the target device. Once /dev/eda1 is located, you will need to format this partition.

Accept the defaults you are given. After a while, you will be asked about installation media. I have chosen NFS, because I am allergic to swapping floppies, but you can try it if you want.

In order to do an NFS installation, you will need an IP address for your machine and the machine with the Slackware disks on it. You will also need to know where the files are located on the mount. If you have mounted a CD containing Slackware on /cdrom with the disks in distributions/slackware, you would export the /cdrom/distributions/slackware directory. In the event you don't have a CD, look forward to downloading. I recommend installing the A and N series for now. It is unnecessary to install any kernels or source, since they won't run on this machine. Go take a break while the installation program is running—brag to your friends about what you are doing, have a pop, etc.

When that step is completed, you are asked to install LILO. Do not do this yet. Follow the normal steps until you are asked if you want to exit. Go ahead. The setup program does not correctly set up the /etc/fstab and /etc/lilo.conf files, so you need to do that yourself. Printed below are the proper commands to type at the shell prompt. Note the append line is the one you first typed with the boot disk.

```
cat <<EOF >/mnt/etc/lilo.conf
        append="ed=cyl,head,sec"
        boot=/dev/eda
        vga=normal
        ramdisk=0
        timeout=50
        prompt
        image=/vmlinuz
                root=/dev/eda1
                label=linux
                read-only
EOF
```

Mount the boot disk you used to start the machine. You can use the /cdrom mount point, like:

```
mount /dev/fd0 /cdrom
```

Copy the kernel from the floppy disk to the hard drive:

```
cp /cdrom/vmlinuz /mnt/vmlinuz
```

The LILO installed by setup does not recognize the major device number 36 that the ESDI drives use, so at this point, get out the disk with esdi_slack.tar.gz on it, put it in the floppy drive and type:

```
cd /mnt
tar zxvf /dev/fd0
```

You will get some error messages, but ignore them. To install LILO, type:

```
lilo -r /mnt -C /etc/lilo.conf
```

The arguments tell LILO the disk with the configuration is mounted at /mnt.

The file system table, /etc/fstab, is still not set up correctly, so you will also need to execute:

```
cat <<EOF >/mnt/etc/fstab
/dev/eda1       /       ext2    defaults 1 1
/dev/eda2       none    swap    swap    0 0
none            /proc   proc    defaults 0 0
EOF
```

At this point the system is ready to be rebooted. Be sure to unmount the floppy and press **CTRL-ALT-DEL**. If the machine does not reboot after about a minute, you will need to cycle the power yourself. Be patient with your system. The ESDI drive is pretty slow. The system should reboot into LILO and start up Linux.

This is still not an exact science. You may have troubles with the partitioning depending on how the drive was formatted before. I've found installing DOS on it will usually make the drive conform to the cylinders, heads and sectors you enter on the append line instead of the physical geometry.

References

David Weis (weisd3458@uni.edu) is a computer science student at the University of Northern Iowa. His favorite things to do include spending time with his girlfriend and solving problems using Linux.

Archive Index Issue Table of Contents

Advanced search

Linux/m68k: Linux on Motorola's 68000 Processor

Chris Lawrence

Issue #37, May 1997

Here's the scoop on the porting of Linux to the Motorola processors.

“Linux is NOT portable (uses 386 task switching etc.), and it probably never will support anything other than AT-hard disks, as that's all I have.” --Linus Torvalds, August 25, 1991.

In the five years since Linus wrote those words, Linux has been ported from its Intel roots to a number of other architectures: the ports to the Alpha and Sparc processors are probably the most familiar to readers of *Linux Journal*. In all the attention given to ports to ever more exotic hardware, it's easy to overlook the first production quality port: Linux/m68k.

The “m68k” stands for the Motorola 68000 series of processors, found at the heart of popular computers like the Apple Macintosh, the Amiga, the Atari ST and its successors (the Atari TT, Medusa and Falcon), as well as the Sun 3, NeXT, Hewlett-Packard/Apollo Domain workstations and others. The MC68020 (with the MC68851 memory management unit), MC68030, MC68040, MC68LC040 and MC68060 are the only CPUs in the 68000 family supported by Linux/m68k, because Linux (like other Unix-like operating systems) requires a memory management unit (MMU) for protected and virtual memory support. A floating point unit is optional, but recommended. Floating point emulation is not distributed in the main kernel tree, since its copyright conflicts with the GNU General Public License.

Like Linux/i386, 4MB of RAM is the absolute minimum, with 8MB being sufficient for most uses. The X Window System requires a minimum of 12MB of RAM for a usable system. A minimal installation currently requires about 55MB of hard drive space, plus at least a few MB of swap space. My personal system currently has about 830MB of hard drive space devoted to Linux (one SCSI hard drive and most of two IDE hard drives). When it comes to RAM and hard drive space, you can never have too much.

Linux/m68k started out as a port of Linux to work only on the Amiga. Hamish Macdonald and Greg Harp released their first version, which they called 0.05, to the public on July 1, 1993. This version was based on Intel Linux 0.99pl9. Soon after that release, several groups of Atari users working independently made the first efforts to adapt the port to that platform. The two ports were merged into one tree starting with 0.9 in July of 1994, with many new features like Ethernet, frame buffer and X Window System support arriving with 0.9pl5 later that year. Further efforts were made to combine some of the advances of the Linux/i386 1.0 and 1.1 series, including ELF support, into the Linux/m68k 0.9 series, culminating in 0.9pl13. Roman Hodek took over maintenance of 0.9 while Hamish started work on catching up with Linux/i386, then approaching the 1.2 release.

The adaptation of Linux/m68k to the general 1.2 kernel was a difficult process. From the first public release (1.2.10), there were 13 patch levels in 11 months (the final release was known as 1.2.13pl10). The format of ext2fs used under Linux/m68k was changed twice (once from 0.9.13 to 1.2.10, and again in 1.2.13pl4). Overall, the 1.2 series saw Linux/m68k mature into a usable system; major improvements were made in X support, and a color display was implemented on the Amiga.

By the time Linux/m68k 1.2 became stable, however, the rest of the Linux community was moving ahead at a rapid pace. Hamish again turned over a usable kernel to Roman and did some very preliminary work on the 1.3 series; Jes Degn Sòrensen adopted the 1.3 source tree in the Autumn of 1995 and began coordinating the work on it. After the initial hurdles of getting the basic code working, progress came quickly. The first working 1.3 series kernel (1.3.23) was released in late February 1996, and was brought into sync by early April (to 1.3.86, one day after the release of Linux).

The current, stable Linux/m68k version is 2.0.28. Development of Linux/m68k continues unabated, with the recent 2.1.17 development release of the main kernel integrating over several hundred kilobytes of changes from the Linux/m68k tree.

As of Linux/m68k 2.0.28, the latest release of the production 2.0 kernel, the Amiga and Atari are directly supported. Users of Motorola VMEbus systems (the MVME 162, 166 and 167) can use an earlier release, 2.0.8. Porting efforts are underway for the Sun 3 and Hewlett-Packard/Apollo Domain workstations and the Apple Macintosh. There has also been some interest in a port to the NeXT workstation.

Compatibility between Linux/m68k and Linux/i386 is very high at the source level. Almost all programs that don't use Intel-specific optimizations (like -

m486), assembler code, SVGAlib or /dev/port should work “out of the box”. Notable exceptions are programs that expect the proc file system's data to be in a specific format (such as /proc/interrupts, which on Linux/m68k can contain any number of interrupts, including shared interrupts). Almost all of the GNU project's software has been tested and used successfully on Linux/m68k, as have the popular Perl, Python and Tcl programming languages and free Web browsers including Arena, Chimera, Grail, Lynx and Mosaic.

As of this writing, no commercial software available for Linux/i386 has been recompiled for Linux/m68k, nor has most other software released without source (with the notable exception of the XForms library). The primary obstacles are as follows:

1. There is no SVGAlib support on Linux/m68k.
2. There is no true Motif port to Linux/m68k. Motif 1.2 has been successfully compiled and used under Linux/m68k, but the individual who did that work doesn't have a license to sell Motif.

Unlike Linux on Intel and Alpha, there is no standard video hardware under Linux/m68k. The Amiga and Atari video chip sets are fundamentally different, as are the various graphics adapters available for both systems. Linux/m68k gets around this problem by using the Universal Framebuffer (UFB) device. This term is misleading, since it is used only on Linux/m68k at this point; but there are plans to merge it with the SparcLinux Framebuffer later. The UFB device abstracts the hardware interface to support a relatively simple, device- and system-independent programming interface. An easy-to-use user-mode library, known as **oFBsis**, is under development as part of the OSIS project to emulate the Atari TOS environment. One side effect of the UFB approach is virtually all Linux/m68k binaries are compatible with all Linux/m68k platforms. For example, the XFree68 server binary can operate all of the display hardware supported by Linux/m68k on both the Atari and Amiga. Even the kernel can be compiled to run on both Ataris and Amigas, and was distributed this way until the 2.0 series, when the number of devices needed for each OS made the combined kernel too large for users with only 4 MB of RAM. More programs supporting the UFB interface are forthcoming.

One of the most exciting developments in recent months is the port of the Debian distribution to Linux/m68k. Debian/m68k is currently in beta testing and will be released in tandem with the next Debian release. Most users currently install Linux/m68k manually using a number of tar files known as the Watchtower-2 file system, a fairly complicated procedure for those not familiar with Unix. There is also an older distribution, based on the 1.2 series kernels, called ALD, available for Ataris on CD-ROM. A proper distribution for both platforms, with support from Amiga and Atari CD-ROM vendors, in addition to

the Linux CD vending community, would help make Linux a viable alternative operating system for serious Amiga and Atari users. At present, the only CD-ROMs available are the ALD CD-ROM and Infomagic's quarterly Linux Developer's Resource 6 CD set.

With the disappearance of the Amiga and Atari commercial developer communities over the past few years, many users have turned to the Free Software Foundation's GNU project for the tools they need. Unfortunately, the Unix heritage of the FSF tools causes problems for Amiga and Atari users who must contend with conflicting file naming formats, weak integration with the underlying OS, and memory-hogging emulation libraries. Linux and other free Unix-like operating systems can provide an environment suited for running these tools, with features like memory protection and virtual memory built-in, at minimal cost.

Substantial progress is underway to run well-behaved Amiga and Atari programs under X. The OSIS project, mentioned above, is usable for many Atari TOS applications already; AmigaOS emulation is also available but slow (via the Un*x Amiga Emulator), with faster support for programs that run within AmigaOS rules being worked on under both UAE and the AmigaOS Replacement OS (AROS). Full-speed Macintosh emulation should also be possible as it is under AmigaOS, but as yet, no one has demonstrated it. Binary compatibility with other Unix-like operating systems on Motorola platforms (similar to iBCS on Intel) is another area that could be developed further and may follow with the Sun 3 port. More emulation support is expected once Linux/m68k becomes easily accessible to Amiga and Atari users and their large freeware authoring communities.

Support for expansion devices under Linux/m68k is rather limited at present. Virtually every Ethernet card ever designed for the Amiga and Atari is supported, but only a relative handful of other devices are supported at present. However, many of them—like the Commodore A2091 and GVP SCSI controllers—are among the most common or—like the SCSI options for the Phase 5 accelerators—the most recent. The relative lack of people with hardware knowledge in the Linux/m68k community has slowed development in this area. With the wider popularity of Linux/m68k that should result after the Debian distribution is released, the dearth of technical expertise should become less of a setback as more people with hardware knowledge join in the development process.

While it is difficult to judge the popularity of other Unix-like operating systems on the Amiga and Atari (primarily NetBSD and OpenBSD), the Linux/m68k Registration Site seems to be a fairly accurate measure of Linux/m68k users. According to the site, well over 400 people use Linux/m68k at least some of the

time. Our Registration Site is prominently advertised at most of the web pages devoted to Linux/m68k, and Geert Uytterhoeven, its maintainer, posts regular messages to the Linux/m68k-related newsgroups with statistics and a registration form. Registrations can be made using a Web-based form at the site, through e-mail or via snail mail. Despite these efforts, many users of Linux/m68k who only occasionally have or do not have Internet access remain unregistered. It is hoped that vendors of Linux/m68k distributions, once they become available, will help publicize the registration site.

The Web has become an increasingly important source of Linux/m68k information. Over a four-day period around Christmas, 350 visits to the primary site of the Linux/m68k Home Pages were recorded. The registration site also receives hundreds of visits per week. The Frequently Asked Questions file and installation guides for Amigas, Ataris and VME systems are available on the Web. Other Linux/m68k pages are available in French, German, Italian and Portuguese. Coupled with the very active developers' mailing list and the Linux/m68k-related newsgroups (in both English and German), users are well-supported with solid information and quick responses from the Linux/m68k user and developer communities.

As most of the developers reside in northern Europe, they have met a couple of times in person. The Linux/m68k project is in many ways a microcosm of the larger Linux project, bringing together people from across the world in pursuit of a common goal. A recent poster to comp.os.linux.m68k commented that the 68000-series processor has many years of life ahead of it. Those of us who work to promote Linux/m68k hope to keep the Motorola 68000 a viable platform for serious computing. With Linux/m68k, you can put together a complete Linux system for well under $1,000. So before you rush out and buy that $8,000 Alpha, dig through your closet, find that old processor, install Linux/m68, and have a computer with the same functionality for a lot less money.

Table 1

Chris Lawrence is a senior Computer Science major at the University of Memphis who has used Linux/m68k on his 68040-based Amiga since February of 1995. He moonlights as a salesman for an Internet service provider, is also known as LordSutch at New Moon (a multi-user dungeon at telnet://

eclipse.cs.pdx.edu:7680/) and mostly leaves kernel hacking to the professionals. He can be reached at quango@themall.net or in care of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Native Linux on the PowerPC

Cort Dougan

Issue #37, May 1997

This quick and non-technical look at Linux on the PowerPC by Cort Dougan, a programmer working on the port, shows it is a reasonable alternative to Linux on the Intel.

Looking for a better operating system for your PowerPC machine? If you're reading *Linux Journal*, chances are you've already found one. The PowerPC-based machines from Apple, Motorola and IBM offer some competition to Intel and DEC Alpha-based PCs. The PowerPC is a well-designed processor with a well-made box, and since it uses the PCI bus, it can use most of the PCI cards made for the Intel PCs. Even better, PowerPCs run Linux.

Where PowerPC Linux Is Now

The PowerPC (PPC) processors are produced by IBM, Apple and Motorola in a joint venture. The vast majority of PPC computers are PowerMacs and PowerMac clones. Both IBM and Motorola make PowerPCs based on the PREP standard. Motorola also makes Mac clones. IBM is now using the PowerPC in its RS6000 and 830 class of machines and in one of its larger computers—the AS400. There is even a version of the IBM portable, the Thinkpad, that is PowerPC-based. The PREP-based machines all look pretty much the same to the operating system, and that makes supporting them all easier. The PowerMac, however, is not PREP-based. Native Linux runs on all of these machines except the AS400 and the older PowerMacs based on the NuBus. Our aim is to be able to run PowerPC Linux on every PowerPC system available.

There are three versions of Native PowerPC Linux, as well as Apple's version of Mach with a Linux personality (MkLinux)--all developed by separate groups. All the native versions of the port started from early work by Gary Thomas. By the time this article is published, we hope the various native versions will have

been merged into one. As things stand right now (in late January), the three versions are:

1. http://www.linuxppc.org/--A version of Linux 2.0 for Be, Motorola and IBM machines by Gary Thomas. This version includes some kernel support from the MkLinux project. More information can be obtained by writing g.thomas@osf.org.
2. http://www.cs.nmt.edu/linuxppc/--A version of Linux 2.1 that runs on the Motorola and IBM workstations from New Mexico Tech. This is the one found in the main Linux source distribution. More information can be obtained by writing cort@cs.nmt.edu.
3. ftp://cap.anu.edu.au/pub/--A version of Linux 2.1 that runs on the PowerMac from Paul Mackerras. More information can be obtained by writing paulus@cs.anu.edu.au.

Currently, we are integrating the New Mexico Tech port (for Motorola and IBM machines) with Paul Mackerras's version (for PowerMacs) in order to support all PowerPC machines from one source tree.

The standard utilities, like Emacs, awk, Perl, bash, Ghostview and TeX, all compile and run just as on any Linux port. Network support is complete—NFS server and client, FTP server and client, slip, ppp, tftp boot servers, xntp and other network services all work. X runs on the IBMs with the S3 card and on the Macs. (It's quite snappy on my IBM 830.) In fact, this document was prepared in its entirety under PowerPC Linux using X, Ghostview, LaTeX, and Emacs.

PowerPC Linux is very stable, and I'm using it as a development platform for the kernel. I've had uptimes as long as two weeks with a reboot only to update to a newer kernel. The infamous “crashme” test has run for over 12 hours on our machine, and improvements are still being made. Lmbench shows the system runs pretty quickly too.

Linux performs very well in comparison with AIX on the PowerPC. It also runs well in comparison to Linux on the Intel when tested with lmbench. The numbers in Listing 1 are the arithmetic mean of the results from 10 runs with lmbench-1. 0 (except on the PowerMac where the results are of a single run). Up-to-date lmbench numbers, crashme results and bug-fixes are kept at http://www.cs.nmt.edu/linuxppc/.

Support for other video cards will be finished shortly. Early versions of shared libraries are working on Thomas's version. They're a high priority, and so will be completed soon. Kernel modules were working up to early versions of the 2.1 kernel, but with the recent changes to the modutils, there are minor problems to be worked out. It will not be long before modules work again.

There is a PowerMac version of LILO (named MILO) that boots MacOS and Linux, but there is no such program for the PREP platforms yet. A boot loader is not yet a high priority, but having one would certainly help in the debugging process—booting new kernels quickly becomes tedious without LILO. LILO is also necessary for users wishing to boot PPC Linux and another operating systems.

Since most PowerPC systems use the PCI bus, PCI cards that work on the Intel PCs work on the PowerPC systems. There are many Linux drivers written for PCI devices, and so adding support for more devices is easy. Most drivers need only minor modifications to work on the PowerPC. Usually we just need to change the “endianness” of the drivers. Most device drivers in Linux assume a little-endian CPU, and since the PowerPC is running big-endian, most drivers need to change the format of the data sent to the device. As changes have been made, the authors have been given copies for inclusion in the standard Linux tree. Luckily, the PowerPC is not the only Linux port to a big-endian machine. The Sparc runs big-endian and must deal with the same issues and fix the same problems, so we're not alone in needing and making these changes.

As of right now, the list of working drivers includes the serial interface, PS/2 mice, EIDE hard drives (CD-ROMs are buggy), the NCR 53c8XX SCSI controller (all SCSI devices working), standard floppy, DEC Ethernet cards (de4x5 driver) and Lance Ethernet cards. Supported on the PowerMac are the MACE Ethernet interface, ADB mouse/keyboard and the MESH and 53C96 SCSI controllers. Mark Scott at Motorola has configured the PowerStack to support audio, but none of the other sound devices on any of the architectures are supported yet. The EIDE CD-ROM support has bugs and needs more work—neither video input nor output works. Patches from users who have other hardware are welcome.

Where PowerPC Linux Is Going

Linux on the PowerPC is a stable and robust development environment. What we need is more users installing it and beginning the work on driver modifications and other missing features. Linux benefits from the work of many programmers across the globe, and PowerPC Linux hopes to have the same advantage.

At this time, making the kernel bullet-proof is the highest priority. Second is speeding it up. After all, a fast kernel that crashes is just a kernel that crashes quickly.

I'd like to take Real-Time Linux, developed here at New Mexico Tech, and make it work on the PPC. The PowerPC makes real-time features easier than the 80x86 with better timer and simpler interrupt interfaces. Integrating with RT-

Linux could even serve to optimize the kernel by using soft disables for interrupts rather than costly hardware disables.

As soon I have access to a symmetric multi-processing (SMP) PowerPC machine, I'll begin work on SMP, since there is no support for PowerPC SMP machines now.

The distribution of PPC Linux currently consists of a boot floppy image for the installation, a root floppy image, a file system tar file and a final boot image for the hard disk. Detailed instructions and the associated files for an installation can be found at ftp://ftp.nmt.edu:/pub/people/cort/. This installation is clumsy and requires a network with an NFS server or a tape drive. This isn't as practical as it could be and leaves much room for improvement. The Red Hat package management tools are compiled and work, but they are not yet directly supported by Red Hat; therefore, only the RPM source packages work.

There is no support in PPC Linux for emulating operating systems other than in the PowerMac version, which runs MkLinux binaries as long as they don't make Mach system calls. Other than limited MkLinux support, there are no plans for adding emulation. Support for PowerPC AIX binaries would not be very difficult, but since there are few applications for PowerPC AIX that users would want, adding support would not be worthwhile. However, a stronger case can be made for emulation of MacOS and Windows. There are many applications for both MacOS and MS Windows that users would want to run under PPC Linux. Perhaps MacOS and Windows emulation for the PowerPC could be taken up by others as a project similar to Wine and DOSemu.

Getting Involved

There is still a lot of work to be done in many areas of the kernel and at the user level. Device drivers need to be modified and tested to translate from a big-endian CPU to the native format of the device. There are very few devices supported now, and I don't have access to them all to do the work. People with hardware they'd like to see supported and an interest in doing some kernel hacking are needed for this project. Even users who don't want to write code can help by testing kernel changes.

People interested in running PowerPC on their workstations are also needed. Different PowerPC machines are needed to test and verify the system works on as many of the PowerPC machines as possible. People willing to help add support for their own machines would be even better.

X needs changes to support more video cards, and the changes should be integrated with standard XFree. I'm rather keen on the idea of a PPC Linux Netscape as well. Linus Torvalds urged the idea of Linux as a “fun” system at

the 1997 Usenix Technical Conference; as an example he cited his work to port Quake to Alpha Linux. Perhaps someone with an interest could take up this cause.

A PowerPC version of LILO that works on the PowerMac and other PowerPC platforms would be very useful. Currently, we only have a PowerMac version, and work on a PREP LILO could begin using the PowerMac version.

Glossary

Cort Dougan is a graduate student at New Mexico Tech and splits his time between his graduate work, PowerPC Linux and hydroponics farming. He can be reached via email at cort@cs.nmt.edu.

Archive Index Issue Table of Contents

Advanced search

Linux? On the Macintosh? With Mach?

Victoria L. Brown

Issue #37, May 1997

Discover MkLinux—what machines it runs on and what it will do when it gets there.

So, Why Linux?

Need you ask? Linux is the overwhelming favorite among users of free Unix clones. Linux provides Unix features such as true multitasking, virtual memory, shared libraries, demand loading, TCP/IP networking and many other advanced features. Versions of Linux have been ported to a wide variety of platforms, including other PowerPC-based computers, making the Power Macintosh port that much easier.

The Linux community is large, growing, active and involved. This community promotes development and exchange of software and ideas, making it an excellent environment for a new OS product. And, last but not least, Linux is covered by the GNU General Public License, ensuring Apple's contributions will not be used in some other vendor's proprietary product.

Why Power Macintosh?

You may be asking yourself, “Why would I want to run Linux on a Power Macintosh?” After all, the Linux community is overwhelmingly oriented toward Intel hardware. Why change?

For one thing, it's good for Apple and Apple enthusiasts. As noted above, Linux opens the door to a new Macintosh market. Many researchers and scientists who might well find the Macintosh a useful tool, cannot justify the purchase of a second computer system. If their shop runs Unix, a Macintosh just didn't fit in—until now.

University laboratories and dorm rooms are another target. With the availability of MkLinux, users can benefit from the best of both worlds: using Linux for research and batch data processing and MacOS for graphical applications, desktop publishing, and much more. So, Apple may sell the machine, but you get the fun. Think of all those cool MacOS applications just waiting to be explored, not to mention the joy of using the Power Mac's multimedia capabilities under MkLinux.

In keeping with Apple's traditions, the highly integrated Power Macintosh hardware greatly eases Linux system administration. Power Macs are delivered as complete systems. Thus, a Power Macintosh normally can run MkLinux straight “out of the box”, without the addition of cards, chips and other components. Because Power Macs use an intelligent bus such as NuBus or PCI, the OS can deal with hardware configuration concerns such as DMA addressing and interrupt vectors.

In fact, as we tell folks at trade shows: “Once you've installed it, MkLinux is really just Linux. You'll have to give up a few things, of course—DMA vectors, IRQ settings, jumpers, incompatible BIOS code—but basically, it's just Linux...”

Although MkLinux, Apple Computer's Microkernel Linux for the Power Macintosh, has been under development for a few years, it has been available to the general public for only a short while. Apple's first public announcement concerning MkLinux was made at the Free Software Foundation's First Conference on Freely Redistributable Software (February 1996).

Apple announced it was supporting a project with the Open Software Foundation (OSF; now merged with X/Open to form the Open Group) to port Linux to a Mach base and to port Mach to a variety of Power Mac products. The project was initiated, sponsored and funded by Apple Computer.

OSF provided the Mach 3.0 Microkernel (developed by Carnegie Mellon University and the OSF Research Institute) and the engineering team to port the code. (An OSF paper on MkLinux—“Linux on the OSF Mach 3 Micro-kernel”—was presented at the conference.)

Apple's February 1996 announcement predicted the first port of MkLinux would become available in the summer of 1996. Exceeding expectations, the first general release of MkLinux, Developer Release 1 (DR1), became available in May. MkLinux DR1 was followed by DR2, released in September 1996. DR3 is scheduled for release in early spring of 1997.

MkLinux releases tend to incorporate large numbers of changes. Hundreds of megabytes of new or changed material must be acquired, whether by FTP or

CD-ROM, typically requiring a complete re-installation. Consequently, full MkLinux releases are made on a relatively infrequent basis (only when warranted by a sufficiently large or fundamental set of changes).

Between releases, Apple issues minor updates via FTP. Some updates provide bug fixes; others introduce new or experimental features. In either case, they are meant to be used with a specific MkLinux release.

What Is MkLinux?

At this point, you may be wondering exactly what MkLinux is. Does it run the MacOS Finder? Does it run X11? Are all the commands I know and love available? For that matter, how is the name itself pronounced?

First things first: MkLinux is officially pronounced “em-kay” Linux, but is often pronounced McLinux. This is in line with Linux tradition, which permits Linux itself to be pronounced in any of several ways. (Li-nucks, Li-nooks, Lie-nooks and even Lee-nooks are quite commonly heard.)

In any event, MkLinux is a complete port of Linux, with a full set of GNU tools and accessories, including X11R6, which runs on top of the Mach micro-kernel. Hence, Mk (Microkernel) Linux. Because MkLinux is really just Linux, it doesn't run the Finder—yet. On the other hand, it *does* run just about any Linux command you could imagine. (Commands that require Intel-based hardware are, of course, impossible.)

A Mach Primer

The Mach Microkernel provides an abstract layer onto which other operating systems can be ported. It also provides multiprocessor support, kernel-level thread support, distributed and cluster computing, and other interesting features. By porting Mach to the Power Macintosh, Apple has cleared the way for a variety of research and even commercial operating systems to run on the platform.

The Mach 3 Microkernel was developed at Carnegie Mellon University. Since then, it has been extensively enhanced by the Research Institute. MkLinux currently uses Mk6.1, a variant of the Mach 3.0 Microkernel, but there are prospects for using a more advanced Microkernel.

The Mach Microkernel performs only a small set of functions. It handles interprocess communication, low-level I/O (that is, access to SCSI and other busses), memory management and scheduling. Higher-level functions (file systems, networking, etc.) are performed by one or more “servers”. Mach servers are user-mode processes that provide all or part of an operating

system's “personality”. They do not talk directly to the underlying hardware; in general, no Mach process does that. Instead, they communicate with the Mach Microkernel by means of “messages”.

Thus, when **cat** performs a **write** system call, the interrupt is caught by the Mach Microkernel. The relevant information is then packaged into a message and passed to the appropriate server. Several actions, interrupts and messages may then take place, involving only the Microkernel and the relevant server(s). Only when the **write** has been accomplished (or fails), does the Microkernel restart the **cat** process.

In MkLinux, as in nearly all Mach-based systems, the OS personality is provided by a “single server”. This is Mach terminology for a single process that handles all of a given operating system's personality. The FSF's Hurd, also based on Mach, uses a “multiple server” design, with a small number of processes sharing the OS duties. The MkLinux project team have received some interest, by the way, in merging the Hurd into MkLinux.

The MkLinux Server

The server MkLinux uses looks much like a standard Linux kernel. In fact, it is a copy of the Linux kernel that is modified to use Mach's low-level functionality. In the first two Developer Releases, the MkLinux server was based on Linux 1.2.13. Updates to DR2, however, as well as the new DR3, are based on Linux 2.0.23. This kernel provides several new features, along with improvements in performance and stability.

Operating system developers will be pleased to know 2.0-based MkLinux allows more than one (e.g., Linux or Hurd) server to run at the same time. This is extremely convenient to anyone who wishes to debug a new server.

With this capability, you can start up the debug version alongside the production version. If (or *when*) the debug version goes down in flames, the system just continues to work, saving you a great deal of time and trouble. Not only that, you can simply fire up gdb and debug the second server as you would any ordinary application.

Linux Goodies

Operating system elitists (read, some kernel hackers) may disagree, but the rest of us know a kernel, however wonderful, isn't enough. We need more: shells, utilities, a window system, and all those other little toys we've grown to love. Don't worry; MkLinux has everything you've come to love in Linux.

Based largely on the Red Hat Linux distribution and making heavy use of the Red Hat Package Manager (RPM), the default MkLinux installation includes a full set of user commands, as well as the complete X11R6 window system. Many other commands are available in RPM archives, either on the installation CD or by anonymous FTP.

In fact, a complete MkLinux system is anything but small or spartan. Even the Developer Releases are quite substantial. (The Installation Guide recommends 16 MB of RAM and at least 500 MB of dedicated disk space.)

GNU… And Apple?!?

We have to admit, Apple and the GNU Project have had their differences in the past. Nor can we suggest Apple has given up on the idea of proprietary software. So, it comes as a surprise to many (and a shock to some) that Apple is openly funding a project to develop MkLinux and port it to the entire family of Power Macintosh systems.

Not only that—except for the chance to sell more Power Macintosh systems (a strong inducement indeed), Apple is not making any profit from the MkLinux port. Distribution and sales of the Apple-endorsed CD-ROM are handled by third parties (e.g., Prime Time Freeware).

In full compliance with the best freeware etiquette, Apple is releasing the source code for all of their Linux and Mach changes under the appropriate (GPL, OSF, and so on) licenses. The entire distribution, in fact, is available via anonymous FTP. Third parties are encouraged to mirror the site, create their own CD-ROM distributions, or share the software with their friends. This is free software at its finest.

What About Intel?

By porting MkLinux to the Power Macintosh, Apple opens the doors for a new market, but that's only half the story. As a bow to the Linux community's Intel orientation, Apple has also made sure an Intel port of MkLinux is available. In fact, the MkLinux/Intel port was developed first and was completely funded by Apple. The Intel port is tracking the Power Macintosh version; although you may not see it prominently displayed on the Apple Web pages, it's still quite alive.

By also porting MkLinux to the Intel platform, Apple opens the door for Intel and Power Macintosh users to try each other's systems, trade software and ideas, and generally enlarge the Linux world. For instance, we expect some valuable and interesting interchange in PCI driver software and multimedia applications.

Nitty Gritty Details

Okay, you're almost convinced. You understand why Apple is funding a Linux project, and you've begun to believe in Linux on the Power Macintosh. But it's still called a Developer Release. How complete is MkLinux? Are all Power Macintoshes supported? What's there, and, perhaps more important, what's not there? The following summary describes the MkLinux DR2 release with all posted updates through early January, 1997.

The first two MkLinux Developer Releases were fairly complete in terms of the base operating system and command set, but were still lacking in a few areas. As noted above, these versions were based on Linux 1.2.13, a somewhat dated version of the Linux server. Linux 2.0 support has recently been added, however (officially, as of the December 1996 update). The MkLinux kernel changes have been sent back to Linus Torvalds for inclusion in the next revisions of Linux; we believe we'll be in sync from now on.

From the beginning, MkLinux has had full SCSI support, including the ability to mount (and eventually boot from) removable disks such as Iomega's Jaz drive. It supports a wide range of monitors connected to the motherboard video or the HPV and A/V cards. It includes serial support for DMA and modem control, plus support for SLIP and PPP connections, as well as Ethernet. X11R6 supports a wide range of multiple-button pointing devices as long as they conform to Apple's Desktop Bus (ADB) protocol.

Several things are still missing, to be sure. Both audio and floppy disk support are still in development. Serial support does not yet extend to printers. At this writing, multiple monitors are not supported; in fact, no NuBus or PCI Bus cards are supported yet. Shared libraries are almost ready; these should be available for Developer Release 3.

To the dismay of many early adopters, MkLinux lacked support for most current Power Macintosh models. DR1 and (as shipped) DR2 supported only the Nubus-based, PowerPC 601-based systems (Power Macintosh 6100, 7100, 8100 series, Power Computing 100 and 120 clones). As these Power Macintosh models were discontinued shortly before MkLinux was first announced, it was impossible for users to buy a new system for MkLinux.

Following the release of DR1, however, the Apple MkLinux Team posted a survey, asking the MkLinux user (and prospective user) community to help choose the next set of machines to be ported. Not surprisingly, the overwhelmingly popular choice was the latest and fastest family of machines—the PCI-bus, PowerPC 604-based chip systems (Power Macintosh 7200, 7500, 8500, 9500, and clones).

Things always take longer than hoped; DR2 was released in September, still without PCI support. We promised support by Christmas, however, and managed to keep our promise. The DR2 update in mid-December contained (beta) support for the aforementioned PCI-based machines, rolled in the 2.0 Linux server, and was a major hit with our long-waiting and patient MkLinux fans.

With PCI support well underway, the team can concentrate on supporting the remaining systems (primarily Performas and Powerbooks) and begin to think about the upcoming CHRP (Common Hardware Reference Platform) systems. The only difficult decision will be which to implement first.

Unfortunately, although many machines seem similar on the surface (and Apple's System Software teams do an excellent job at making them look the same!), they're really all a little bit different inside. So, it may take a while... but rest assured, the team is committed to making MkLinux available on all of the Power Macintosh platforms in time.

The History and the Team

MkLinux was started as the dream of Brett Halle, then manager of Apple's kernel team within the Modern OS department. With the blessings of Apple Vice President Ike Nassi, Brett began sponsoring a handful of OSF Research Institute employees to port the Mach 3.0 Microkernel, and Linux, to the first Power Macintosh platform. Several months into the project, the first Apple engineer, Michael Burg, came on board to work part-time on the MkLinux effort.

Shortly before the DR1 release, Apple decided the project was worth a little more backing and spun the two Apple employees (Halle and Burg) off into their own, dedicated team. What became Apple's Leveraged Technologies Group is now up to five employees, with three more engineers at the Research Institute and hopes for reasonable growth in the future.

Unfortunately for our anxious and growing body of MkLinux fans, this is still a very small team. While we concentrate on porting to the next series of Power Macintoshes, keeping our Web pages and FTP site up to date, and managing the whole project, many interesting developments are “resource-limited”. Fortunately, this is Linux, where “everything is done by someone else”. The MkLinux Developer's Corner is a small but intrepid band of MkLinux programmers who are willing to take on (and complete) needed projects. Our Developers Corner has provided us with the X11R6 port, NetaTalk, GNU-step for MkLinux, HFS filesystem utilities, and a number of other interesting and desirable additions. We're happy to count these developers as members of the MkLinux team.

Last, but not least, our thanks go to all the MkLinux users who bravely download and install each new update as it is posted. In a small internal project such as MkLinux, we don't have access to Apple's dedicated software testing organizations. We've tried to test and debug our Developer Releases and updates before they are released, but we rely on our user community to stress-test our releases in a wide range of network environments and hardware configurations. We've been most impressed by the helpful comments, willingness to get involved, cogent bug reports and sensible e-mail we've received from all these folks.

You Can Join Our Team!

The MkLinux team currently numbers over 15 registered developers, some 4000 registered users and 5000 mailing list subscribers. (We admit some of the mailing list subscribers are also registered users.) If you haven't joined our team, we'd be happy to welcome you.

The current release of MkLinux is always available by both anonymous FTP and on CD-ROM. Our FTP site, ftp://ftp.mklinux.apple.com/pub/, is mirrored by nearly two dozen sites worldwide. The Apple-endorsed CD-ROM, emblazoned with MkPenguin (the Linux penguin, sitting on a Power Mac), is available via mail order from the publisher, Prime Time Freeware (http://www.ptf.com/, info@ptf.com).

Due to their experience in Unix and freeware publishing, Prime Time Freeware has been selected to publish Apple's reference release of MkLinux. Edited with the assistance and support of the Apple MkLinux team, *MkLinux: Microkernel Linux for the Power Macintosh* will contain both a tutorial introduction to MkLinux and a variety of interesting and useful reference material. By the time this article goes to press, the reference release should be in print.

Composed of a book and two CD-ROMs, the product will contain a variety of reference material about Linux, Mach, MkLinux, and the Power Macintosh. *MkLinux: Microkernel Linux for the Power Macintosh* is the only reference work for MkLinux, containing a variety of material that will be unavailable from any other single source. It will be available in many technical and professional bookstores and by direct mail order from the publisher.

Visit our web site (www.mklinux.apple.com) and look around, then join some mailing lists. We strongly recommend you join mklinux-announce and mklinux-answers; these are moderated lists (low in volume, high in relevant information) keep users abreast of important events in the MkLinux community. The remaining (topical) groups provide a means for you to interact

with other MkLinux developers and users, sharing ideas, problems, and solutions. See you on the Net.

MkLinux Combatability List

Vicki Brown has been working with Unix systems of one sort or another since 1983, much of that time in the employ of Apple Computer. Currently a member of the MkLinux project team, she describes her job duties as Firewarden, Web Gardener and Stagehand. In her spare time Vicki enjoys reading, keeping up with Star Trek and *Babylon 5*, and spending time with her spouse and four cats. She can be reached at vlb@apple.com.

Archive Index Issue Table of Contents

Advanced search

Tcl/Tk with C for Image Processing

Siome K. Goldenstein

Issue #37, May 1997

See how to use a mix of Tcl, Tk, and C to make image manipulation both easy and efficient.

To start an implementation in C from scratch for an image processing (or manipulation) program is a difficult task. It is necessary not only to develop an internal data structure, but also to write the filters for reading and writing the available graphic formats. The interface design and implementation is also difficult, due to the need for dealing with issues such as color allocation, quantization and so on. In this article, we'll show you how Tcl and Tk can help you in dealing with these problems easily. However, it should be noted that some operations on images are computationally intensive, making the use of Tcl prohibitively expensive. So we'll use a mixture of Tcl and Tk with C, and get the best of both worlds.

In *Linux Journal* #10 (February, 1995) Matt Welsh wrote a nice article describing a way to use Tcl/Tk as a front end for C programs using pipes to and from a wish process. While this method has many advantages, e.g., straightforward implementation and memory saving when using static libraries, it does present some limitations:

- First, since your program is “split” into two different processes, the sharing of resources is not an easy task.
- Second, all communication is done through the pipes, imposing an extra burden on the system.

In this article, we approach this problem using Tcl/Tk as an extension to the core program, and show some of the advantages of solving it in this manner.

A Practical Example: Let's Dither

We'll start by writing a small program to do a special dither (half-toning) for creating a special effect that applies only to a selected sub-rectangle of an image.

The described technique transforms vertical strips of colored pixels into a vertical strip of black and white pixels, where the average intensity best approximates the original average. Also, all black pixels are grouped in the center. (This effect has been used in the entertainment industry for some time now.) See Figure 1. The following sections describe the necessary steps for accomplishing this effect.

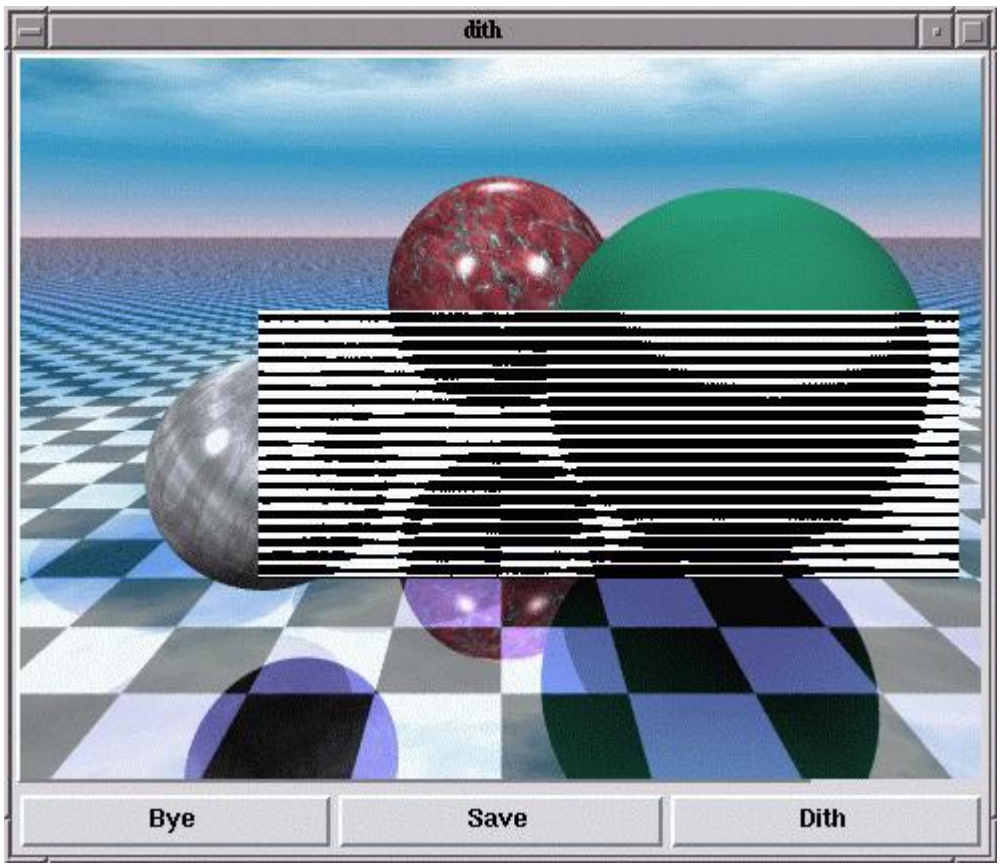


Figure 1. The typical appearance of the program after the dithering of one rectangular region.

Image in Tk

The very fabric of our program is based on the **image** primitive, which first appeared in Tk version 4.0. The idea is to create an “image object” with an associated command, just like any normal widget.

Images can be of two different types: bitmaps and photos. While bitmaps deal only with foreground and background colors, photos treat true-color objects, automatically quantizing for the available number of colors in the display. Let's

focus on the “photo” type, which was implemented by Paul Mackerras based on his earlier “photo widget”.

A command for creating an image object named “picture” with the image in the file “mickey.gif” would be:

```
image create photo picture -file mickey.gif
```

After its creation we can easily do some operations. For example, to get its dimensions:

```
set pic_wid [image width picture]
set pic_hei [image height picture]
```

You can also create a second image, and copy a section of the first one into the second:

```
image create photo pic_piece
pic_piece copy picture -from 0 0
        [expr $pic_wid/2] [expr $pic_hei/2]
```

During the copy you can use the Tk options **subsample** or **zoom** to reduce the image or enlarge a portion of it. The copied portion can be placed anywhere inside the destination image.

It is possible to specify the size of the color cube of a given image (you can even explicitly impose it to be gray-scale), its gamma correction and some other nifty things. Check out the **photo** man page for details.

A good way to both see the image and allow some manipulation is to treat it as a “canvas object”:

```
canvas .c
pack .c
 .c create image 1 1 -anchor nw -image picture
                -tags "myimage"
```

After creation, you can draw and manipulate any canvas object you wish just as if it were floating upon **myimage**. Just remember to keep the image as the “lower” object, so that you'll always be able to see everything else. This positioning can be accomplished by giving:

```
.canvas lower myimage
```

Tcl/Tk as an Interface for Your C Programs

Let's make a small distinction between two kinds of C-Tcl/Tk applications: those which act like a shell (wish, for example) and those which use the Tcl/Tk extension in a predetermined way.

If you want to create another “instance” of wish with some extra commands you have created, you should read the man pages concerning **Tcl_Main** and **Tcl_AppInit**.

If your program uses Tcl/Tk only for the interface, and it is not intended to be used in a “shell-like” fashion, the approach is slightly different. I recommend you grab the nice demo program tkHelloWorld.tar.gz (see Sidebar) to use as an example.

Basically, your program has to implement the following four steps:

- Initialize Tcl and Tk.
- Create the Tcl commands responsible for calling your C routines.
- Ask Tcl to evaluate an “interface description” file.
- Let Tk control the main flow of the program.

In the C code shown in Listing 1, the comments identify exactly which of the four steps is being done.

Listing 1

From this point on, we wish to use C programming only for some critical functions, since the main flow and control of our application is handled by Tk.

Calling C Functions from Tcl

If you are interested in the myriad ways you can call a C routine, read *TCL and the TK Toolkit* by John K. Ousterhout, Addison-Wesley, 1994.

Essentially your C function must have a prototype like the following:

```
int
C_func_name (ClientData cd, Tcl_Interp *interp,
             int argc, char **argv);
```

and you must register it by:

```
Tcl_CreateCommand (interp, "Tcl_func_name",
        C_func_name, (ClientData *) NULL,
        (Tcl_CmdDeleteProc *) NULL);
```

Then, whenever Tcl encounters the command **Tcl_func_name**, it will call your routine, which will receive the Tcl parameters just as **main** receives the **argc** and **argv** arguments from the shell, i.e., **argc** will be the number of words and **argv** will be the “vector of strings”.

Passing Images Back and Forth

We want our C routine to process an image called **image_name** under Tk. The immediate solution would be to pass the color of each pixel (the photo widget has this option) again and again until the image is complete. While this program was running, we could go out for lunch, visit a few friends, have dinner and see a movie. However, there is a better way to accomplish the goal. From C, we ask Tk to take care of it. First, we have to define:

```
Tk_PhotoHandle      image;
Tk_PhotoImageBlock  blimage;
```

Then call the following functions in sequence:

```
image = Tk_FindPhoto ("image_name");
Tk_PhotoGetImage (image, &blimage);
```

The image is in **blimage**, which is a structure defined in tk.h as:

```
typedef struct {
  unsigned char *pixelPtr;
  int width;
  int height;
  int pitch;
  int pixelSize;
  int offset[3];
} Tk_PhotoImageBlock;
```

All color information comes in unsigned characters (values between 0 and 255). The **pixelPtr** is the address of the first pixel (top-left corner). The **width** and **height** define the image dimensions, while **pixelSize** is the address difference between two horizontally adjacent pixels, and **pitch** is the address difference between two vertically adjacent ones. Finally, the **offset** array contains the offsets from the address of a pixel to the addresses of the bytes containing the red, green and blue components.

Using the above definitions allows different representations of the image; for example:

- Define a point with a dimension of three bytes, one for each color component. Then the **pixelSize** is 3, the **offset** 0, 1 and 2 and the **pitch** three times the width.
- Think of the color image as three planes (images), one for each color. Then the **pixelSize** is 1, the **offset** is 0, **width*height** and **2*width*height**. Finally, the **pitch** is equal to the width.

The colors of a given pixel can be obtained with three simple C macros:

```
#define RED(p,x,y)  ((p)->pixelPtr[(y)*(p)->
pitch + (x)*(p)->pixelSize + (p)->offset[0]] )
#define GREEN(p,x,y) ((p)->pixelPtr[(y)*(p)->
pitch + (x)*(p)->pixelSize + (p)->offset[1]])
```

```
#define BLUE(p,x,y)  ((p)->pixelPtr[(y)*(p)->
pitch + (x)*(p)->pixelSize + (p)->offset[2]])
```

You call the macros giving the address of the block structure explained above as the first parameter, and the x and y coordinates (where 0,0 is the upper-left corner) of the pixel as the second and third. For an optimized program, it would be much faster to use address differences to determine the position of the next pixel from the current pixel, i.e., its neighbor.

About the Program

The complete C code for this program is in Listing 1, and the Tcl code is in Listing 2.

Figure 2 is a snapshot of the program in action.

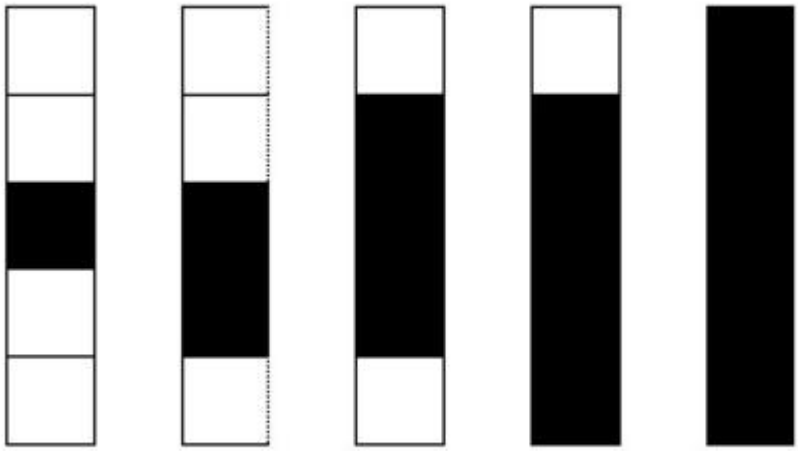


Figure 2. An Example of all possible output clusters, when the vertical size is five. The criterion of choice is the nearest average.

Listing 2

The program can be downloaded from: ftp://ftp.impa.br/pub/visgraf/people/siome/lj/ljdither.tgz.

An Important Remark about C and Tcl/Tk Interaction

When Tcl/Tk calls a function in C, it can still receive interface events, such as button presses or slider movements; however, it cannot run the associated scripts (or C functions) bound to these events, since for the moment the C function controls the flow.

A good example is a mass-spring simulator, where the C function has a loop doing the simulation and canvas drawing. It would be wonderful to be able to change the constants during the simulation, or even abort it before the pre-determined time. This option is also needed in long Tcl scripts. The solution in both cases is to use the **update** command from time to time in order to process user input.

From the **update** man page:

> The update command with no options is useful in scripts where you are performing a long-running computation but you still want the application to respond to user interactions; if you occasionally call update, user input will be processed during the next call to update.

Conclusions

A powerful combination is achieved by letting Tcl/Tk deal with the interface and C with the critical tasks of a program.

A lot of useful extra widgets can be found on the Internet for using sound (see tkSound), moving objects and so on. The principle for integration of these widgets is the same—you can create a new wish-like shell, or use the new available functions together with come extra C code of your own.

Another good package is Tix, which is included with many Linux distributions. It adds many wonderful widgets to Tk, and has an object-oriented approach to building new “mega-widgets”.

I hope you find this article useful, and have a nice hack.

Resources

Siome Goldenstein is an Electronics Engineer who is currently finishing a Masters degree in Computer Graphics. He loves Aikido and non-technical reading. Siome lives in Rio de Janeiro, Brazil. Comments can be sent to him via e-mail at siome@visgraf.impa.br.

Archive Index Issue Table of Contents

Advanced search

Internet Servers in Perl

Mike Mull

Issue #37, May 1997

In a sequel to his “Perl and Sockets” article in the March 1997 issue of *Linux Journal*, Mike Mull demonstrates how Perl can be used for the server end of a socket connection.

In my previous article in Issue #35 of *Linux Journal*, I wrote about the socket library functions in Perl with an emphasis on writing Internet client programs. Perl is also a good language for Internet servers, not only because of the socket capabilities and the ease of dealing with files and data, but because it also has a special mode for improving security. In this article I cover several aspects of writing Perl servers, including how to use the basic socket functions, how to best handle multiple connections, asynchronous communication and security issues. In the process we'll develop a simple Internet server similar to fingerd that works through the Web.

Socket communication may be either connection-oriented or connectionless. Connection-oriented protocols, like the Internet's Transmission Control Protocol (TCP), establish a link between client and server before exchanging any data. Connectionless protocols, like the User Datagram Protocol (UDP), simply read or write data, specifying the client or server address each time. Most servers use a connection-oriented scheme, and we use this approach in our example server (see Listing 1). However, I discuss the connectionless approach below.

Any Internet server, from the simplest to the most complicated, first uses the two functions **socket** and **bind** to establish an identifiable communications endpoint. The server uses **socket** to create a socket with the desired type and protocol. Recall the syntax for this function is:

```
socket SOCKET, DOMAIN, TYPE, PROTOCOL
```

SOCKET here is a Perl file handle initialized by the call to socket. For Internet TCP applications **DOMAIN** is **AF_INET** and **TYPE** is **SOCK_STREAM**. The Perl 5 Socket package defines the constants **AF_INET** and **SOCK_STREAM** as well as other socket-related constants and functions; refer to the previous article for details. The

An Internet server must bind a network address to the socket with the **bind** function. A client can bind an address, but it is not usually necessary in connection-oriented clients. This is also referred to as “naming the socket”. This process specifies the network address to which a client must connect to start communicating with the server. The syntax of **bind** is:

```
    bind SOCKET, NAME
```

The **SOCKET** argument is still the file handle created by the call to **socket**. **NAME** is the address that is bound to the socket. The contents of this argument can be quite complicated (again, refer to the previous article for details). For versions of Perl from 5.2 on, a function in the Socket package called **sockaddr_in** returns a value for the **NAME** argument given a port number and an Internet host address. If you're writing something like an ftp or HTTP server, you can use the reserved “well-known” port number (see the file /etc/services for these numbers). Otherwise, any positive 16-bit integer will suffice as long as it is not one of the reserved numbers. For servers the special argument **INADDR_ANY** can be used for the Internet address, which lets the kernel pick an address for the socket.

For connection-oriented servers like our example program we now can use the **listen** function to tell the operating system that we'll accept connections on the socket. This function looks like this:

```
    listen SOCKET, QUEUESIZE
```

We all know what **SOCKET** is by now. **QUEUESIZE** specifies the number of attempted connections that can be kept waiting; the symbol **SOMAXCONN** is the maximum for this argument (usually 5). This lets the server handle several near-simultaneous connection requests, a crucial feature for HTTP servers or daemons like inetd.

Now a client program could attempt to connect to the server, but we need more code to actually create the link. For many servers, the **accept** function is called, typically in a loop of some sort, directly after **listen**. The syntax of **accept** is:

```
    accept NEWSOCKET, GENERICSOCKET
```

This function opens **NEWSOCKET**, a file handle that you can read from or write to in order to communicate with the connecting client. **GENERICSOCKET** is any open, named socket. For our server, this is the named socket we've already created with **socket** and **bind**. **accept** returns the address of **NEWSOCKET** in the same form as the **NAME** argument to **bind**.

Note that the **accept** call waits until a connection request arrives, so no processing can occur until it completes. This usually poses no problem since it matches the way most servers work: they wait for a request and then service it. Sometimes, though, an application might perform other tasks, like calculation or system monitoring, that can't be stopped to wait for client connections. If so, communication can be done asynchronously—that is, processing can be interrupted temporarily using a signal handler to make the socket connection and to process the client's request. I don't cover this in detail since that requires a lengthy digression into the **fcntl** system call and signal handlers, but Listing 2 illustrates the basic idea.

UDP does not guarantee reliability; extra user code must deal with problems caused by packets that don't make it to their destinations. The Internet's main connectionless protocol is called UDP, or User Datagram Protocol. A **datagram** contains all of the information required to send it to the right place. needed. For a connectionless server, **listen** and **accept** are not needed. A connectionless client usually does need to use **bind** so that a valid return address gets passed to the server in the client's data packets, but we won't worry about the client side here. To use UDP on our socket rather than TCP, we simply replace the **socket** argument **SOCK_STREAM** with **SOCK_DGRAM** and the **getprotobyname** argument **tcp** with **udp**.

In C we use the system functions **sendto** and **recvfrom** to send data between client and server with UDP, but Perl doesn't implement these directly. Instead, Perl uses **send** and **recv** for both connection-oriented and connectionless protocols. After setting up the socket with **socket** and **bind**, a connectionless server would usually call **recv**:

```
recv SOCKET, SCALAR, LEN, FLAGS
```

This function blocks until data becomes available on **SOCKET**, then reads LEN bytes into the scalar variable **SCALAR**. **FLAGS** are the same flags as for the **recv** system call. **recv** returns the address of the client, which can then be used to send information back with the **send** function:

```
send SOCKET, MSG, FLAGS, TO
```

TO is the client address. The socket code in the simplest connectionless server would look something like this:

```
socket(S, AF_INET, SOCK_DGRAM, \
   getprotobyname('udp'));
bind(S, sockaddr_in( $port, INADDR_ANY) );
$cli_addr = recv S, $request, 80, 0;
send S, $message, 0, $cli_addr;
```

Now back to our TCP server. Remember I mentioned earlier that several connection requests can get queued up so the server can respond to each in turn. This might be inefficient (and probably annoying to the client user) if the server does something that takes a significant amount of time, like querying a database or running an external program. To get around this problem, many servers fork a new process to handle a request once they accept a connection. Look at our example server code for details. The only slightly tricky part is the **CHLD** signal handler used to clean up zombie processes.

Servers often run as setuid or setgid programs, meaning the processes have the privileges of the user or group that owns the executable file regardless of who runs the program. At the very least, a server program will run under your own user ID. Since anyone can, in principle, use an Internet server, you can see security is of the utmost importance. You must make sure the server does not give privileged access to important system files or your own confidential data. Usually this requires checks on environment variables, file privileges, external program execution, etc., so that it's hard to be thorough. Fortunately, Perl helps us out here with its *taint mode*, a mode that checks for common security violations. The **-T** command line switch turns on this mode, so we just add this to the “shebang” line at the top of the script.

The **exec** function in the example server might cause security concerns for at least two reasons. First, executing an external program implies the use of the **PATH** environment variable. This variable is considered to be tainted until we set it explicitly in the script, since it could be modified to cause the execution of a program other than the one we intended. Second, we separate the arguments to **exec** into the program name and the argument list, which prevents **exec** from calling the shell to do metacharacter substitutions. If these modification were not made, the taint mode would send warnings to the terminal and stop the program (in fact, that's how I found these problems). Keep in mind taint mode does not guarantee security, but it does make it much easier to identify well-known problems.

Network servers are among the most complex pieces of software, which is to say, you should by no means consider this article a comprehensive treatment of the subject. Still, you'll be surprised to find how many of the elements of our simple example program show up in even large, complicated servers. Perl does reduce some of the complexity though, since you already have convenient tools at hand to do the hard parts, like parsing protocols and manipulating files. Even if you ultimately decide to write the program in C or some other compiled

language, Perl can't be surpassed for prototyping server applications. The price is right too, but I don't need to convince Linux users of the value of “free” software.

Mike Mull writes software to simulate sub-microscopic objects. Stranger still, people pay him to do this. Mike thinks Linux is nifty. His favorite programming project is his 2-year-old son, Nathan. Mike can be reached at mwm@cts.com.

Archive Index Issue Table of Contents

Advanced search

An Interview with DEC

David Rusling

Jon Hall

Issue #37, May 1997

David Rusling and Jon “maddog” Hall talk about Digital Equipment Corporation and the porting of Linux to the 64-bit Alpha.

The Alpha port of Linux actually started on two fronts, one in the Littleton, Massachusetts offices of Digital Equipment Corporation, and one on a riverboat in New Orleans, Louisiana.

The first front was started by Jim Paradis of Digital Semiconductor's Alpha Migration Tools group. Jim's group is responsible for finding innovative new ways of using Alpha processors, and Jim had been looking at several versions of the Unix operating system to determine if there was a possibility of doing a port. Jim had determined that Linux would give an efficient, powerful operating system, and he proceeded to start a 32-bit port of Linux to the Alpha as a test case. Jim believed that a 32-bit version of Linux would be the easiest platform for porting the GNU tools, the X Window System and other applications.

In the meantime, Jon “maddog” Hall was helping to plan for a DECUS event in New Orleans. DECUS is the Digital Equipment Corporation Users' Society, and the Chairman of the DECUS Unix Special Interest Group (Unix SIG), Kurt Reisler, wanted to bring Linus Torvalds to DECUS to speak about Linux. This was in May of 1994, and V1.0 of the kernel had only recently been released. Kurt was having trouble funding Linus's trip from Helsinki, so the Digital UNIX Base Product Marketing Team funded the trip. On meeting Linus, maddog was as impressed with the young man as he already was with the operating system he had designed. After DECUS, while riding the riverboat *Natchez*, the prospect of doing a 64-bit port to the Alpha was broached, and a short time later, an Alpha system was on its way to Helsinki.

maddog:

About that same time, I found some of the Digital UNIX engineers were using Linux. Through contacts made in the engineering group, I located Jim Paradis and opened up conversations with him. Shortly afterward, a small team of engineers were put together to do further work on the Alpha port of Linux. Through marketing research, I convinced the team that joining their efforts with Linus in doing his 64-bit port was the thing to do. Although the 32-bit port was further along at the time, the 64-bit port was moving fast, and the issues around 32/64-bit porting were not materializing to the extent Jim had expected. Therefore, Jim “packed up” the 32-bit port, and the engineering staff started to attack the 64-bit port along with Linus and several brave volunteers.

David Rusling:

When this project started, Linux only ran on Intel x86-based systems. The early days involved a lot of frantic activity tracking Linux kernel releases, and at the same time, porting Linux to the Alpha AXP platforms. The earliest port of Linux to the Alpha AXP was based upon the 32-bit Intel kernel, and in retrospect, maybe we should have been braver and gone straight to 64-bit native AXP support. At that time Linus was working toward exactly this end. His approach was to port the kernel to AXP and run Digital Unix binaries to get a working system. Our approach was to take 32-bit Intel Linux and port that and the rest of the system applications (for example **init** and **bash**) to AXP without disturbing their natural 32-bit-ness. We swapped a lot of code with Linus but in the end felt we were not contributing to the mainstream effort—which was always Linus's codestream. I guess there was just so much to be done that all of the work was useful, but in the end, we switched over to Linus's 64-bit AXP kernel work and contributed there. Very little of the early work was wasted; the work I did on MILO (the Alpha Linux Miniloader) ported straight over to 64-bit Linux, as did Jay Estabrook's device driver work and Jim Paradis's memory management and systems work. The PCI code used in all of Linux's platforms also comes from that time period. Our first goal was to get a self-hosting Alpha Linux system that did not crash and had working SCSI and Ethernet device drivers. We needed SCSI to support the file system and Ethernet device drivers to support networking. There had to be enough applications to be able to build the Alpha Linux kernel.

In the very early days, there were no other Alpha AXP users besides the three of us in Digital and Linus. Another hardy early contributor was David Mosberger-Tang, who was brave enough to buy an Alpha AXP Noname system and start hacking. The five of us worked hard and the code flowed freely. Our early goal was to have a free distribution of Alpha Linux that could be taken and used by the greater Linux Community. Jim Paradis's BLADE release contained enough of a system for people to easily get Alpha Linux up and running on their system

well enough to start writing code. At the time BLADE came out, there were only two widely available supported Alpha Linux systems, the Jensen (DECPC 150) and the Noname board (so called because any clone vendor could put their name on it). The Noname board was reasonably—although still rather expensively—priced, and quite a few were purchased for running Linux. BLADE got Alpha out to the mainstream Linux community—or, at least, to the braver souls in the Linux community.

maddog:

Another market research "coup" was the decision to have only one source code tree for the kernel, as was the determination to have the Alpha version of Linux as close to the Intel version as possible. One of the early projects based on this decision was to move away from the Extended Common Object File Format (ECOFF) and to do Executable and Linking Format (ELF) instead. ELF had already been selected for the Intel side, and the Intel version of Linux had to transcend the a.out-to-ELF formats. The ELF project started immediately, and the transition from ECOFF to ELF happened before Alpha Linux rolled into common usage.

David Rusling:

Before ELF, the image sizes were enormous since every image was statically linked. When ELF came along, the entire system usage got a whole lot better. The change was particularly dramatic on the DEC Multia/UDB (Universal Desktop Box); you could run X in 32MB of memory and *not* use any swap space.

maddog:

The third big marketing decision was that Digital would not sell Linux, but instead allow the existing vendors of Linux systems to supply the distributions and support. This led to the decision to have Digital engineering contribute back in source code (whenever legally possible) everything they had developed for Linux.

David Rusling:

At Digital we never saw ourselves as "owning" Linux; Linux cannot be owned or controlled by anything but the worldwide Linux community. Throughout the early days we were essentially playing catch-up with Intel Linux. We were writing code, porting code and running around showing people this neat, new thing. We Digital folks saw (and see) ourselves as part of the wider Linux community. I had not even used Linux before MILO first booted it on an Alpha. Linux is infectious. Our role was, and still is, to be a catalyst to the rest of the Linux world. A good example is the Red Hat Alpha Linux release. We helped

make that possible, but it was Red Hat who did the real porting work. They did it because, like us, they believe Alpha AXP is a good Linux platform. Another example of Digital leveraging the larger Linux world is the driver for our TGA (or Zlxpe) graphics adapter. This card had no XFree86 support but was being used in our Windows NT and Digital Unix Alpha systems. Jay Estabrook did a port of XFree86 for that device, and we also released TGA sources to the XFree86 project.

maddog:

Many might ask why the Alpha port is important. First of all, the Alpha is the world's fastest single-chip microprocessor. Linus wanted a fast system to show what Linux could do, and the Alpha was a natural choice.

Secondly, the Alpha is a true 64-bit system, with 64-bit integers and a 64-bit virtual address space. This allowed Linus to see what the kernel would require in the way of page tables, etc. On another plane, the large address space of the Alpha allows Linux to be used in research for computing large memory models —particularly interesting in parallel systems and clustering techniques.

Third, the Alpha is a very clean RISC architecture. In the beginning, it was so clean that the Alpha did word accesses only to memory, which forced some of the kernel data structures to be redesigned for efficiency. Later Alpha chips have byte instructions, but this cleanliness of the architecture helped to set the pace for other RISC architectures.

David Rusling:

Many folks have taken to the Alpha because they like its technology. Its purity of architecture, speed and natural 64-bit support are very attractive to technologists. There was a lot of interest from the research and academic communities—the combination of an operating system that we already knew and loved, together with falling prices of Alpha-based systems helped sales quite a bit.

maddog:

Fourth was the fact that Digital was an early adaptor of the PCI bus. Although some early Alpha systems had Turbo-channel buses for backwards compatibility with our older systems, Digital started early in the Alpha system life to provide PCI support, at the same time blending in EISA and ISA support, too. This made it easier to move device drivers from Intel Linux to Alpha Linux, which would probably have been more difficult if the systems had a proprietary bus, such as the Sbus or NuBus.

David Rusling:

Alpha Linux has been able to capitalize on two things. Firstly, Alpha PCs with their PCI and ISA buses can use exactly the same devices as Intel PCs. Secondly, the price of an Alpha PC is mostly determined by its component prices and, as the price of the Alpha Linux OS has fallen, Alpha PCs have correspondingly become much cheaper, and thereby, more attractive.

maddog:

Fifth was the fact that Digital was encouraging clone makers by selling Alpha systems, boards and chips to original equipment manufacturers who wished to make their own configured system boxes. In order to make this as inexpensive and open as possible, the Alpha Linux engineering team chose the Advanced RISC Computing (ARC) console systems (commonly used with Windows NT), and wrote a source-code-available version of the console code (called MILO) as well as a source-code-available version of the privileged architecture library (PAL) code.

But from a general viewpoint, it was important that a 64-bit port be done, and done with a different architecture than Intel. At the time the Alpha port was started, the Linux kernel was highly optimized for the Intel architecture, utilizing more than a little Intel assembly language and Intel architecture tricks in places like context switching, and it was only a 32-bit port. Likewise, the source code tree and the kernel structure were not optimized for various architectures. By porting to the Alpha, the Linux team paved the way for all the other ports to be completed and integrated cleanly.

David Rusling:

There is more to Linux than the operating system. Most of the useful functions of Linux are in the system utilities that make the Linux operating system what it is. Bash, X servers and so on, are what the user sees and uses. The kernel is a relatively small (but important) part of the system. As the small but determined group of Alpha Linux users grew, they concentrated on getting more of these system utilities to run. Meanwhile, at Digital we concentrated on supporting the hardware platforms and devices we believed would make good Linux systems. For example, Digital Semiconductor sells a range of PCI network chips under the codename “Tulip” (the 21x4x device range). A lot of Linux device drivers were not particularly portable then, since some had embedded Intel assembly code in them as well as 32 bit-isms. Jim and Jay spent a lot of their time porting working Intel Linux device drivers over to Alpha Linux, ironing out portability issues. maddog was waving the Alpha Linux flag at various trade shows, and in August 1995, we were able to show Alpha Linux running an S3 X server at the Unix Expo. I worked on the Alpha Linux Miniloader, MILO, which is, roughly

speaking, LILO on acid. It is a freeware loader that allows Windows NT Alpha boards to boot and run Linux. It uses real Linux device drivers and file systems to get the Linux kernel loaded and running. As soon as Jim and Jay got the device drivers working in Linux, I'd get them working in MILO. MILO lowers the entry price of Alpha Linux as you do not have to pay either the system resource monitor (SRM) console license fee or the Digital Unix license fee to buy a hardware platform that runs Linux. In the spirit of Linux and the Free Software Foundation, MILO is freely available and redistributable code.

maddog:

At this point the Alpha project is about at parity with the single Intel processors from a functionality standpoint. Work is starting on systematic multi-processing (SMP) for the Alpha, and the things that are missing for the Alpha tend to be missing (for the most part) from the Intel Linux systems as well—applications. The Java port that is being worked on for Alpha Linux will certainly help fill this gap, as will projects like GNUstep and other application environment issues.

David Rusling:

Today Linux's multi-architectural support is very natural. Almost everything a Linux user or system needs can be built for Alpha straight out of the box. Our early aim was to make Alpha Linux an attractive alternative to Intel Linux. With off-the-shelf Linux distributions, most device drivers running happily and a wide choice of supported graphics devices, we have achieved our goals. Along the way we have had fun, made friends and learned new stuff. What is left to do for Alpha AXP Linux? First, we will continue to support our hardware and, as we build new Alpha PC motherboards, we will either port Linux ourselves or make the information available that will allow others to do the port. Second, we will make technical information available to the wider Linux community. The one thing I have learned about the Linux community is that there are a lot of bright and able programmers who only need the right information and some access to the hardware to do an excellent job. Third, we need to make Alpha Linux systems price competitive with Intel Linux systems. When we were first porting Linux, the price differential was very high, but in the last two years, that price gap has closed significantly due to more companies cloning and selling Alpha PC systems in volume. Linux is responsible for a fair proportion of that volume.

maddog:

We could use some more tuning of the compiler optimization sequences which the gcc compilers generate for the super-scalar Alpha architecture. Likewise, certain math libraries need to be optimized and made available in source code format, not only for the Alpha, but for other ports as well.

We would like to see a virtual porting and certification lab on the Internet, so applications developers who do not have Alpha systems can port and test their applications. This would also be a good idea for some of the other ports, such as SunSPARC, PowerPC, etc.

Testing and benchmarking of Alpha systems running Linux under different load types, creating meaningful benchmark results would also be useful.

Doing real work in large-scale distributed computing with “clusters” of Linux systems would also provide helpful information.

Also needed is a defined set of applications program interfaces (APIs) and application binary interfaces (ABIs) that fit across a variety of Alpha Unix and Linux systems (FreeBSD, netBSD, Linux, Digital UNIX and a variety of other Unix systems) so that commercial application vendors could create shrink-wrapped applications for a larger audience than any one Unix system could attract. Applications tested against the ABI should be able to run on any Alpha Unix/ Linux system.

David Rusling:

I agree. The future of Linux (all flavours) rests on its ability to attract applications. Whilst the normal engineering set of tools (Emacs, LaTeX, Tk and so on) works quite happily on all of the Linux platforms, Linux needs more of the marketing and presentation tools. It needs a viable desktop environment. That can either be the ability to run Windows applications via Wabi or it can be native applications conforming to some interface specification. The free software world is unfortunately less interested in WYSIWYG applications than in writing operating systems.

One option I find really attractive is the idea that Java applications could run under Linux as well as, if not better than, any other operating system incorporating a Java Machine.

maddog:

While Digital has allocated four engineers, one product manager and one very over-worked marketing manager to the task, we realize none of this could be possible without the long hours contributed by the Linux community.

We want to help the community move Linux along the path that they feel is the best.

David Rusling is Principal Engineer of European Semiconductor Applications Engineering, Digital Equipment Co. Ltd.

Jon “maddog” Hall is Senior Leader of Digital UNIX Base Product Marketing, Digital Equipment Corporation

Archive Index Issue Table of Contents

Advanced search

Safely Running Programs as root

Phil Hughes

Issue #37, May 1997

Every time you are running as root, you are taking a chance. With a little programming, you can decrease the need to be root and make your life a little safer.

This article is more about ending a bad habit than serious programming. How many of you regularly become root to do some routine task? I thought so. And, worse yet, how many of you just stay logged on as root because you know you can do what you want? That's too many.

One very common reason to become root is to run a shell script needing root privileges. For example, starting PPP is usually done by a script that must be run as root. For this article, I use that example as the basis for the code shown in Listing 1. There is nothing special about it—it just happens to be a common example. This same program can be modified to run other scripts requiring root privileges or to do other root-like tasks.

The first thing you need to understand is the meaning of the phrase “set UID on execution”. This concept is the only patented feature in Unix. It is the ability to “look like another user” while executing a program. The most common example is running the **passwd** program to change your password. If you look at the permissions on the password file you will probably see something like this:

```
-rw-r--r-- 1 root  1260
Nov  3 10:05 /etc/passwd
```

Notice that only root has permission to write to the file. Now look at the permissions on the password program:

```
-rwsr-xr-x 1 root 10636 Jun 6 1996
/usr/bin/passwd
```

Notice there is an **s** where you would expect an **x** to indicate execute permission for the owner. The **s** indicates the “set UID” bit is set.

Having the UID bit set means when you, as an ordinary user, execute the passwd program, the program is executed as if you were root. This enables you to change your password entry in /etc/passwd, but you won't be able to do anything else. The program itself (/usr/bin/passwd) is responsible for making sure you do only reasonable tasks; since you don't have write permission to the program, you can't change it.

If you understand set UID, you can now see how important it is in guaranteeing program security. For example, if your program has a way to get into the shell, it has a security hole.

While we are talking about security holes, one other approach is allowing shell scripts to run set UID. This ability actually exists in some Unix systems, and it opens huge security holes. Ideally, you must be able to read the script and trust it.

The program I wrote to start and stop PPP is in Listing 1. Its purpose is execution of the appropriate shell script to start or stop PPP, depending on whether it is called with an argument of **on** or **off**.

Most of the code is explained in the comments, but let me further explain a couple of items. First, I chose to set the PATH environment variable to a reasonable set of directories. It is important to do this to guarantee it's impossible to sneak an unauthorized executable into the program. Second, I used the **execle** system call to execute the appropriate script. **execle** passes the new environment to the called program, so it inherits the search path I set instead of the path of the calling user.

I also specified the full path name of the programs to run (see the #define lines)—another security consideration. It should be unnecessary after PATH has been set, but it's an inexpensive safety precaution.

Finally, the program must be installed properly. Once you have built the executable (**make ppp**), you should become root, move it to an appropriate directory (e.g., /usr/local/bin) and correctly set the permissions.

If you want anyone to be able to run the program, make sure it is owned by root and set the permissions to 4711. The leading 4 specifies setting the set UID bit. If you have a particular group of people you want to allow to run the program, change the group owner of the program to the appropriate group with permissions set to 4710.

That's it. If all goes well, you now have one less excuse to work logged in as root.

If you need a system for allocating various root tasks to ordinary users, solutions are readily available. Check out the **sudo** and **super** programs, which are included in most Linux distributions.

Phil Hughes is publisher of *Linux Journal*. In a past life he was a Unix systems programmer.

Archive Index Issue Table of Contents

Advanced search

LJ Interviews Przemek Klosowski

Marjorie Richardson

Lydia Kinata

Przemek Klosowski

Issue #37, May 1997

Mr. Klosowski tells us about his users group and why it is such a success.

Marjorie Richardson, Editor of *Linux Journal*, and Lydia Kinata, SSC Products Specialist, interviewed Przemek Klosowski, the founder of the highly successful Washington DC Linux Users Group. The interview was conducted via e-mail on January 21, 1997.

Marjorie: First, why don't you tell us a little about yourself?

Przemek: I was born in Poland, and I lived there until 1985. I came to the US to attend graduate school in Physics, at the University of Notre Dame. I now work as a researcher at University of Maryland. Besides work, I enjoy being with my wife, daughter and cat. We like nature walks, alpine skiing and socializing (we have been known to have 8-hour meals with our friends). My music tastes are rather eclectic—I like Bach as well as Bartok, Miles Davis and the band "Morphine". I also like tinkering with things mechanical, electronics and computers. Therefore, my favorite radio programs are "Schickele Mix" and "Car Talk", conveniently broadcast back-to-back on our local public radio station.

Marjorie: Tell us something about your club's history: when you got started, number of charter members, etc.?

Przemek: We started the club in July 1994. There are pockets of computer industry around the Beltway, including the suburbs north of Washington where I live. Hoping to reach such computer folks, I posted announcements to some mailing lists and newsgroups, and distributed leaflets in libraries and computer stores in the area. I expected a few people to show up for the first meeting, but

my expectations were exceeded—over 70 people appeared. Since then, we have had about 30 people attend regularly, with some meetings attracting over a hundred folks. Our most popular meetings featured a WWW talk, when the Web was still a novelty, and a talk by Linus Torvalds himself.

Our DCLUG (DC metropolitan area Linux User Group) is the oldest and largest club, but other groups also serve people in farther regions of the DC urban sprawl—LUGMAN in Manassas, Northern Virginia, and UMLUG (University of Maryland folks). In addition to our Linux activities, we are friends with the Washington Area Unix User Group (WAUUG) and the DC System Administrators' Guild (DC-SAGE). We are cooperating on talks and activities; quite a few people are active in multiple groups.

Marjorie: How many members do you have now and what are the group's demographics (i.e., age, gender, profession, etc.)?

Przemek: We have a varied distribution: a bunch of Gen-Xers, and then a regiment of computer industry veterans—people who actually remember the beginnings of the computer revolution. I do worry that we have very few students; this is partly due to the fact that we aren't meeting anywhere near a school, meaning a long commute for the typical student.

Indeed, selecting a meeting place is very important. An ideal place would be near major highways *and* public transportation; a university campus might be a good idea. We meet in an auditorium at the National Institutes of Health—a nice place, but it's quite a commute for most of us.

As for gender, unfortunately we don't have many female members; we would certainly welcome more gender and ethnic variety, but the simple fact remains that the typical member is a young male.

Marjorie: Tell us about a typical meeting. Are they fairly informal, or do you have planned talks and activities?

Przemek: We normally schedule a speaker for our meetings. I believe it helps to keep up with the effort. If nothing in particular is scheduled, it is somehow easier to stay home. Featuring scheduled talks keeps the momentum up.

Marjorie: Do you host special training sessions and/or install fests? Any annual events?

Przemek: David Lesher and David Niemi did a wonderful job organizing our four Install Fests. Interestingly, our last fest had a small turnout compared to the previous ones—could it be that Linux has become so easy to install?

Actually, both Davids agree that the trickiest part in organizing the install fest is the preparation—choosing the right time, spreading the word and registration. We insist on registration not only to have a handle on what to prepare in the way of resources and equipment (desk space, network hookups, etc.) but, perhaps more importantly, to get the participants to collect solid information about their hardware.

Marjorie: Do you attend conferences as a group and set up a booth? If so, which conferences? Any experiences you'd like to share with us?

Przemek: We have attended the Washington DC conference (FedUnix/Open Systems World) for three years in a row; the organizer (Alan Fedders of the Washington Area Unix User Group) kindly lets us set up a booth there. We usually shift our November meeting to coincide with the conference place and time; this way, we can reach more people who might not hear about us otherwise.

We run the club with a minimal amount of organization. We do not have formal membership, club dues or formal leadership. This makes it more difficult to operate events. We can't rent space and/or equipment, and we look like poor cousins of better-endowed clubs such as PC User Group (the DOS/Windows folks) or even the area Unix user group. On the other hand, none of us want the club to develop into a part-time job, so we don't mind.

We haven't found the lack of funds to be a limitation—we get by through trading favors, borrowing things, doing the work ourselves and light fund-raising.

One coup we managed to pull off is our very own Linux server. David Niemi made arrangements with Erol's, a large ISP in our area, to let us use one of their computers connected to their internal backbone. This allows us to maintain a local high bandwidth Linux FTP mirror as well as our own WWW server (http://linux.wauug.org, with our club's home page at /dclinux/dclinux.html) and assorted mailing lists. Needless to say, the machine runs under Linux. In return, we try to help Erol's in various little ways.

For fund-raising, we distributed some vendor CDs during our install fest (we obtained them on commission, so we didn't have any up-front costs). The income will enable us to expand disk space on our server in the near future.

Marjorie: What are some of the reasons your members give for using Linux?

Przemek: I haven't asked widely, but my guess is that people use Linux because it provides a “turn-key” development platform, often with the capabilities not

provided by commercial systems (Unix or Windows), and of course, you can't beat the price. Actually, for many people it is not the money that matters most. Rather, it is the aggravation of completing and configuring their system from many disjointed elements (a TCP stack from this company, an NFS server from that company, an X server from a third one, to give a Wintel-related example)--and that was just for basic system functionality. Then, one still has to load all the software that comes standard with Linux distributions: compiler tool-chain, multi-media/WWW tools, etc.

For those of us who have to develop solutions to nonstandard computer problems, as is often the case in my area of scientific research, Linux offers a better platform because of its openness and freedom. A perfect example is the adaptation of Linux to do hard real-time embedded computing (described recently in *Linux Journal*)--such a project would be impractical without the availability of the kernel source.

And, last but not least, Linux lets us (some say “forces us to”) hone our computer skills. I have learned many things about computers (hardware and software) while using my Linux system.

Lydia: How many of your members are primarily using Linux in their business? What applications do they use?

Przemek: Quite a few of our members use Linux at work, but in many cases it is low-key and/or their bosses don't know (or want to know) about it. There are some spectacular exceptions, e.g., Donald Becker uses Linux as the basis for the Beowulf project (http://cesdis.gsfc.nasa.gov/linux/beowulf/beowulf1.html). And Erol's uses Linux extensively as an Internet server platform, as do other ISPs in the area.

In general, people in science, research and development tend to have an easier time applying Linux—there is not as much enforced uniformity. For instance, in my laboratory, many researchers have their own Intel-based PCs. Some prefer Windows, some prefer Linux. Linux tends to be easier to maintain—once set to working, it stays working.

For those who require Microsoft applications in their work, we often set up Wabi. This gives us the best of both worlds: good, user-friendly applications, on a solid OS. If only the anti-trust law enforcement people would actually do their job and break up Microsoft into an applications company and an OS company, everyone would be much happier.

At the moment, we use Linux to do mundane things—graphics display, word processing, data visualization—but we plan to employ it for actual control of

hardware instruments. In fact, if some young reader is interested in such topics and likes living near DC, please get in touch with me; I am looking for an able young programmer.

Lydia: Do you or any of your members contribute to the development of the Linux kernel? Utilities or patches?

Przemek: Donald Becker wrote a majority of the network drivers for Linux. Eric Youngdale worked on iBCS, Intel and Alpha-shared libraries and high-level SCSI code. David Niemi co-maintains Mtools, and has worked on the floppy driver and “other bits here and there”.

Others among us have contributed to various parts of the kernel and applications, in both big ways and small (I contributed in minor ways to various projects such as Emacs Calc, xmgr and tcl/expect).

Lydia: What are your thoughts on the pros and cons of the currently available distributions?

Przemek: I personally like Red Hat, because housekeeping on their system is easy due to the Red Hat Package Manager (RPM). Upgrading the Red Hat system is easier than other systems that I have used. Other people swear by Yggdrasil, Debian, Craftworks and Slackware—the differences aren't that great.

Marjorie: What do you see as the main purpose of your users group? Users groups in general?

Przemek: Linux user groups let people create their own local support systems. This is important especially when an answer can't be found by searching the available documentation. True, everyone can post questions to worldwide Linux newsgroups, but getting an actual answer is a different matter. A local support market is more friendly, too.

Marjorie: To what do you attribute the success of your group?

Przemek: I think half of being successful is to stay in business long enough to create a distinct group of people willing to participate over the long term.

Marjorie: What do you think the future holds for Linux and Linux user groups?

Przemek: Linux is here to stay and flourish. It is a complicated system, and every once in a while it requires some expertise. The nice thing is that if you have a problem, you *know* it is solvable, and will usually yield to the “scientific”

method of experimenting and progressive elimination. This is to be contrasted with the “let's install the latest version of a screen driver” approach.

Linux user groups concentrate local Linux knowledge and are therefore a valuable resource. I imagine businesses using Linux might appreciate such a resource and draw on it for consulting and/or hiring, although we aren't at this stage yet.

Lydia: How far do you think Linux can go as a competitor for Microsoft operating systems or commercial versions of Unix?

Przemek: I don't think Linux will get a “market share” comparable to that of Microsoft Windows. I propose that even if there were a free, high-performance engine which fit the Ford Escort, most people would still drive the factory-provided one for it. Similarly, most people would probably opt to trade off the competitive features of Linux, because they don't want to, or cannot, cope with its complexity.

I do think Linux will continue to have a major impact on the computer industry by shaming it into action. It must be embarrassing for vendors to fail to provide some functionality (a network protocol or a hardware driver or protected virtual memory)--it can no longer be brushed off as “too hard to do”.

As to commercial Unix, I think Linux will entrench and become a major player. As of today, in our lab we have more systems running Linux than all other commercial Unices combined.

Thank you for the opportunity to talk about our club and about Linux.

Greetings,

Przemek Klosowski (przemek@nist.gov)

Python Update

Andrew Kuchling

Issue #37, May 1997

Python has evolved since we last had an article on it. Andrew Kuchling brings us up to date in this article, and we invite readers to submit suggestions for Python topics Andrew might cover in future issues of *Linux Journal*.

What's been happening to Python since J. Bauer's article in *Linux Journal* #35? Like most free software, Python is being continually developed and enhanced. At the time of the original article, Python was at version number 1.2, and betas of 1.3 were floating around. Since then, version 1.3 has been officially released, only to be replaced by 1.4 in late October.

Versions 1.3 and 1.4 have both added new features to the language. The really significant new item in 1.3 was the addition of keyword arguments to functions, similar to Modula-3's. For example, if we have the function definition:

```
def curse(subject="seven large chickens",
          verb="redecorate",
          object="rumpus room"):
    print "May", subject, verb, "your", object
```

then the following calls are all legal:

```
curse()
curse('a spaniel', 'pour yogurt on', 'hamburger')
curse(object='garage')
curse('the silent majority', object='Honda')
```

Arguments not preceded by a keyword are passed in the usual fashion; non-keyword and keyword arguments can be used in the same function call, as long as the non-keyword parameters precede the keyword parameters. By that rule, the following call is a syntax error:

```
curse(object='psychoanalyst', 'a ancient philosopher')
```

and the following call would cause an error at runtime, because an argument is being defined twice:

```
curse('the silent majority', subject='Honda')
```

As a pleasant side effect, adding keyword arguments required optimising function calls, reducing the overhead of a single function call by roughly 20%.

Most of the changes in the 1.4 release made Python more useful for numeric tasks. Many of the changes were proposed by the members of the Matrix special interest group (or Matrix-SIG), which has defined a syntax and built a data type for manipulating matrices. (The Python SIGs are small groups of people tightly focused on one application of Python, such as numeric programming or database interfaces; see http://www.python.org/sigs/ for more information about the existing SIGs.)

One such enhancement is support for complex numbers. The imaginary component of a complex number is denoted by a suffix of “J” or “j”; thus, the square root of -1 is symbolized as 1j . The usual mathematical operations such as addition and multiplication can be performed on complex numbers, of course.

```
>>> 1+2j*2
(1+4j)
>>> (1+2j)*2
(2+4j)
>>> (1+2j)/(2+1j)
(0.8+0.6j)
```

The presence of complex numbers also requires mathematical functions that can perform operations on them. Instead of updating the existing math module, a new module called cmath was added; old software might malfunction if an operation returns a complex value where an error was expected. So math.sqrt(-1) will always raise a ValueError exception, while cmath.sqrt(-1) will return a complex result of 1j.

```
>>> import cmath
>>> cmath.sqrt(-1)
1j
>>> a=cmath.log(1+2j)
>>> print a
(0.804718956217+1.10714871779j)
>>> cmath.exp(a)
(1+2j)
```

For the sake of users comfortable with Fortran's notation, the **** operator has been added for computing powers; it's simply a shorthand for Python's existing pow() function. For example, 10**2 is equivalent to pow(10,2) and returns 100.

One minor new function has been requested by several people in comp.lang.python. Python has long had a tuple() function which converts a sequence type (like a string or a list) into a tuple; the usual idiom for converting sequence types to lists was map(None, L). (The function map(F,S) returns a list containing the result of function F, performed on each of the elements of the

sequence S. If F is None, as in this case, then no operation is performed on the elements, beyond placing them in a list.)

Many people found this asymmetry—tuple() existed, but not list()—annoying. In 1.4, the list() function was added, which is symmetric to tuple().

```
>>> tuple([1,2,3])
(1, 2, 3)
>>> list( (1,2,3,4) )
[1, 2, 3, 4]
```

An experimental feature was included in 1.4 and caused quite a bit of controversy: private data belonging to an instance of a class is a little more private. An example will help to explain the effect of the change. Consider the following class:

```
class A:
    def __init__(self):
        self.__value=0
    def get(self): return self.__value
    def set(self, newval): self.__value=newval
```

Python doesn't support private data in classes, except by convention. The usual convention is private variables have names that start with at least one underscore. However, users of a class can disregard this and access the private value anyway. For example:

```
>>> instance=A()
>>> dir(instance)  # List all the attributes of the instance
['__value']
>>> instance.get()
0
>>> instance.__value=5
>>> instance.get()
5
```

A more significant problem; let's say you know nothing about A's implementation and try to create a subclass of A which adds a new method that uses a private __value attribute of its own. The two uses of the name will collide. Things are slightly different in 1.4:

```
>>> instance=A()
>>> dir(instance)
['_A__value']
```

Where did this new value come from? In 1.4, any attribute that begins with two underscores is changed to have **_** and the class name prefixed to it. Let's say you have a class called **File**, and one method refers to a private variable called **__mode** the name will be changed to **_File__mode**.

```
>>> instance.get()
0
>>> instance.__value=5
>>> instance.get()
0
>>> dir(instance)
['_A__value', '__value']
```

Now, this still doesn't provide ironclad data hiding; callers can just refer explicitly to **_A__value**. However, subclasses of A can no longer accidentally stomp on private variables belonging to A.

This feature is still controversial and caused much debate in comp.lang.python when it was introduced. Thus, its status is only experimental, and it might be removed in a future version of the language, so it would be unwise to rely on it.

Both the 1.3 and 1.4 releases included some new modules as part of the Python library, and bug fixes and revisions to existing modules in the library. Most of these changes are only of interest to people who've written code for earlier versions of those modules; see the file Misc/NEWS in the Python source distribution for all the details. If you're just coming to the language, these changes aren't really of interest to you.

The news isn't just limited to the software. The first two books on Python were published in October: *Programming Python*, by Mark Lutz, *Internet Programming with Python*, by Aaron Watters, Guido van Rossum, and James C. Ahlstrom. At least one more book is scheduled for release next year.

Two Python workshops have taken place, one at the Lawrence Livermore National Labs in California last May, and another in Washington, D.C. in November. Speakers discussed all sorts of topics: distributed objects; interfacing C++ and Python, or Fortran and Python; and Web programming. See http://www.python.org/workshops/ for more information about the workshops and the papers presented.

In November 1996, the 5th Python Workshop was held, in association with the FedUnix '96 trade show. The two most common topics were numeric programming and Web-related programming. For numeric work, there's a lot of interest in using Python as a glue language to control mathematical function libraries written in Fortran or C++. Code can be developed quickly in Python, and once the program logic is correct it can be ported to a compiled language for speed's sake. There's also a benefit from using a general programming language like Python, instead of a specialized mathematical language; it's easier to make the numeric code accessible with a GUI written in Tk, or with a CGI interface.

Another popular topic was Web-related programming. The Python Object Publisher was an especially interesting system, which enables accessing Python objects via HTTP. To take an example from the Object Publisher presentation, a URL like:

http://www.here.com/Car/Pinto/purchase?name=Bob

causes a Python Car object named Pinto to be located, and its purchase() method will be called with 'Bob' as a parameter. Other presentations discussed generating HTML, writing tools for system administration, and collaborative document processing. Brief notes on the papers are at http://www.python.org/workshops/1996-11/, with links to HTML or PostScript versions.

As you read this, plans for the next workshop are probably in progress, though there's no news at the time of writing; see the Python Web site for the current status. In the past, the meetings have alternated between the Eastern and Western U.S., so workshop #6 will probably be on the West Coast.

References

Archive Index Issue Table of Contents

Advanced search

FairCom's c-tree Plus

Nick Xidis

Issue #37, May 1997

c-tree Plus will work on almost every platform with a C compiler and applications written on one platform can quickly and easily be ported to others.

- Manufacturer: FairCom
- Price: $895; additional same site programmer licenses may be purchased for 40% off retail price.
- Reviewer: Nick Xidis

Industrial strength cross-platform btree file management.

c-tree Plus is a professional developer's package. It has a complex set of features and can produce commercial grade multi-user applications. It will work on almost every platform with a C compiler and applications written on one platform can quickly and easily be ported to others. It can even work around most of the platforms' native file handling weaknesses. If you aren't a professional programmer odds are that you may have never heard of FairCom or c-tree Plus. The manual says:

> c-tree Plus is a sophisticated file management product designed to be used in a variety of operating system environments. Written in the C language it is designed to be integrated with your C application program to provide easy to use, yet flexible data file manipulation and indexing.

Wow, what a mouthful! Let's cover the basics first. For those of us who don't eat and sleep C programming, a simple, and therefore good, way to think of c-tree Plus is as a library of C functions that manage data files and indices similar to the way Borland's dBase worked on old DOS platforms. You'll see later that this is professional development package with a lot more to offer than DOS dBase, but bear with me.

The Basics

In the simplest application, like a business's database of customers, a single data file can be accessed through multiple binary tree (btree) index files. Each time your application accesses the data through a different index, your data looks like another data file sorted by the index you are using. You may create one index on the customer's name and another on the customer's account number. Looking though the customer name index the data appears to be sorted by customer name. Use the account number index and through the magic of btree indexing the same data appears to be a new data file sorted by account number.

How to Use ISAM with c-tree Plus

Let's run through an outline of how a simple application is built. c-tree Plus offers a full suite of Indexed Sequential Access Method (ISAM) functions that can perform multiple file and index operations in a single ISAM function call. The low-level c-tree Plus functions are also available if you really need to have complete control, but the author and FairCom recommend that you stick with the ISAM functions. In order to be fully compatible across a wide variety of platforms, c-tree Plus uses it own data types. For example, an “int” may be a two-byte integer on one system and a four byte integer another. With c-tree Plus you use COUNT, which is always a two-byte signed integer.

C-tree also provides a pointer to its custom types in the form of pCOUNT, which is the equivalent to COUNT *. One of the most important custom data types is the IFIL, which defines the parameters for an ISAM file instances. You can declare your data structure in your sources or use separate parameter files. I like using the structures in my sources because when you open and close files individually with ISAM parameter files it's all or nothing.

The IFIL structure is pretty straightforward—it contains a pointer to the file name, a file number, the data record length and pointer to an IIDX index file structure. Next you'll declare the parameters for the index keys within the data records for each instance of the data using an IIDX structure containing a key length, key type and flags if duplicate keys or null keys are allowed, and a pointer to the index file name. The ability to have duplicate keys is an important feature of the c-tree package. When you use duplicate keys, the c-tree system actually pads each to make it unique. Your application doesn't have to deal with any of it—it's all taken care of for you.

The last structure is an ISEG which tells c-tree where to find the key values in data records via a segment length and offset. That's it! Just three basic structures and your data schema is laid out. The functions to work with the file

structure are well thought out and very intuitive. CreateFile, OpenFile, AddRecord, GetRecord and DeleteRecord all do exactly what you'd expect.

A Mode for Every Occasion

Now a few words about modes. There are four c-tree modes, two single-user and two multi-user. The features vary between each but the base functionality is the same. The simplest, single-user mode, gives you get all the features except file security. The other mode adds Online Transaction Processing (OLTP). C-tree's OLTP suite is awesome; you can roll back transactions, use all kinds of save points, and enjoy wonderful logging and easy recovery. In the author's biased opinion, OLTP is really a must for mission-critical business applications—this package does it right.

Now for the multi-user modes, where the horsepower of this package really shines. First is the non-client/server mode. It uses the traditional methods for file and record locking to which Unix programmers are accustomed. Using Linux's NFS capabilities you could use this mode to create a client/server application, but you'll give up OLTP and a lot of c-tree's file buffering. We compiled some simple applications and found that this mode quickly slowed as you added users. For small businesses where OLTP would not unduly hold things up, using NFS and multi-user mode may be a very cost-effective solution for shared data applicants.

Now for the crown jewel of the c-tree suite: client/server mode. The package includes one FairCom server binary for the platform you specify (Linux, OS/2, Netware, DOS, NT, etc.); this is the only portion of the package for which source code is not available. I was surprised to find there are servers for just about every platform—yes, even lowly DOS can run the FairCom server and handle record locking and file security.

On multi-tasking platforms such as Linux, the server runs in the background and is very well-behaved. When you use client/server mode you get the full set of OLTP, file security, and file and record locks. The FairCom server seems to be fast, but I don't have the facilities to really load it up. Your applications communicate with the server via TCP/IP sockets, shared memory or message queues. I ran it using TCP/IP over Ethernet and PPP dial-up and found the performance to be very good. Another neat trick is that in client/server mode you're able to transparently access the local disk with or without OLTP at the same time. For the most part, you can change modes by recompiling your applications with the right c-tree libraries for that mode. This is very cool, since you can go from a single-user OLTP application to a client/server application by recompiling with a different c-tree library.

The Installation

The installation is a snap—just follow the Installation & Quick Start Guide. First, run **tar -xvf /dev/fd0** for each of the two Unix floppies that come in the box. Then run **tar -xvf ctreeX.tar** to unpack the sources—yes, you get the full c-tree Plus source code. FairCom provides its own make programs to walk you through the build process. It's in this area that FairCom could do a little better. I found the process of compiling the make program—running the **mtmake** configurations utility which sets up the compiler flags, then running FairCom's make program, **mk**--a little tedious. Especially when you have to repeat the last steps three times if you want the stand-alone, multi-user and client/server mode libraries. In addition you may want to edit the **ctree.mak** file by hand and add your favorite compiler flags. I added **-O** and **-Wall**--yes, I'm paranoid and like to see lots of warning messages. On the plus side their **make** programs will work the same on any platform.

Conclusion

FairCom's c-tree Plus is a proven winner that's been around for a long time (since 1979). It's a wonderful base to build serious business applications. And with FairCom's per developer, not per platform, license a Linux system c-tree is a very cost-effective, cross-platform development tool.

The Linux community needs more professional development tools like this. If you are a professional or “wanna be” professional developer looking to build mission critical business applications, I highly recommend that you take a look at c-tree Plus and its report generator and 4GL companions r-tree and d-tree. You can get more information at http://www.faircom.com/.

Archive Index Issue Table of Contents

Advanced search

Relinking a Multi-Page Web Document

Jim Weirich

Issue #37, May 1997

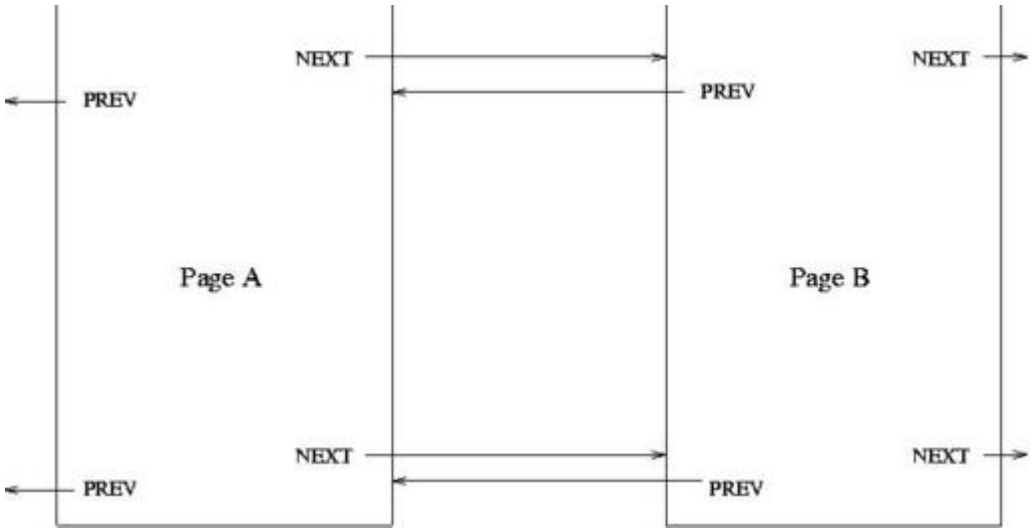
When you need some help getting your web pages back in order, have the computer do it for you.

There is something magical about writing a web-based document that just doesn't exist with a regular linear document. Something about getting all those links just right and in the right sequence makes a web document come alive. Of course, getting the links just right can be a big job, especially in a document with many pages. I found that out when I tackled my first multi-page document.

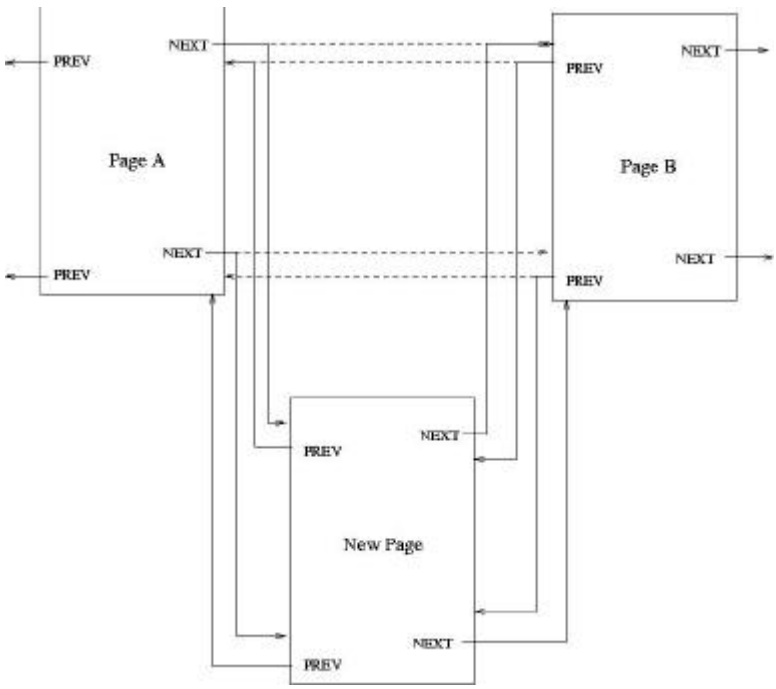
I had been writing HTML for several months when an opportunity came to make a presentation at our local Internet Special Interest Group (part of a larger PC users group). At that time, only a few of us were “on the Net”, but many people were interested in what the Internet—and more specifically—what the Web could do for them. I volunteered to give a talk on the basics of HTML and putting together your own web page.

The group met in the library of a local university, and we had a live Internet connection tied into an overhead projector in the room. I decided it would be neat to write a presentation about HTML in HTML. Each web page would be a single slide in the presentation. Links between pages would allow me to move forward (and backward) as the talk progressed.

So I put together about 15 pages of slides and linked them so each page had a *next* link to the next page and a *prev* link to the previous page. I put these links at the top and bottom of each page, so there were four links on every page (actually, I had links to the table of contents too, but let's ignore those for the moment). Figure 1 shows how consecutive pages are linked.



The talk went well, but I saw several places where I could improve the talk. When I started adding pages to the document, I made a very important discovery: inserting pages was a big pain. If I wanted to insert a new page between existing pages A and B. I had to update the NEXT links in page A, update the PREV links in page B, and update both the NEXT and PREV links in the new page. And because I had links at the top and bottom of the pages, there were twice as many links to update. Figure 2 shows the revised links.



Automation to the Rescue

After struggling with manual updates to the pages, I decided there had to be a better way. The **relink** Perl script was a result of that frustration.

Using **relink** is simple. First you need a file (called **links**) containing a list of pages in the order they are to be visited. Omit the ".html" portion of the page name in the **links** file, **relink** assumes the files end with that extension.

For example, consider the following (very abbreviated) version of my original HTML presentation. I start with an introduction (**intro.html**), have a page about anchors (**anchor.html**) and finish with a conclusion (**conclude.html**). The **links** file would contain:

```
# Pages for a simple presentation
intro
anchor
conclude
```

Each HTML page contains a set of links to its next and previous page. For example, the **anchor.html** file contains the following links at the top and bottom of the page.

```
<a href="conclude.html">
   <img src="icons/next.gif" alt="NEXT">
</a>
<a href="intro.html">
   <img src="icons/prev.gif" alt="PREV">
</a>
```

After reviewing my short document, I feel that I really should mention URLs and how they work before delving into anchors. So I write a new page called **url.html** and wish to add it to my document. I simply edit the **links** file to contain:

```
# Pages for an updated, but still
# simple presentation
intro
url
anchor
conclude
```

After running **relink** with the new page order, the links in the anchor page will now look like:

```
<a href="conclude.html">
   <img src="icons/next.gif" alt="NEXT">
</a>
<a href="url.html">
   <img src="icons/prev.gif" alt="PREV">
</a>
```

Notice the previous link now points to the page about URLs, rather than the introduction. The links in the other pages are updated in a consistent manner to support the new page order. Pages can be added, deleted, or simply rearranged just by editing the **links** file and specifying the new order.

Identifying Links

How does **relink** find the HTML links in a web page? It does so by looking for particular patterns on lines containing a hypertext link. **relink** will scan through an HTML file looking for the pattern **/href\s*=/i** which matches the letters **href** followed by zero or more spaces followed by an equal sign. The **i** at the end of the pattern allows matching without regard to upper and lower case. Lines

matching this pattern contain a hypertext link and are possible candidates for updating.

Once a line containing a link is found, a list of link-specific patterns is tested against that line. If a match is found, that hypertext link is updated with information obtained from the **links** file, and the scanning process continues on the rest of the file. For this process to work, it is important that each hypertext link fit alone on a single line of text. Also, link-specific patterns must be chosen that do not occur normally in the body of the document. If a link-specific pattern should accidently appear on the same line as an unrelated link in the document body, **relink** will automatically (and incorrectly) update that unrelated link.

I use small GIF files for the next and previous icons, so the link-specific patterns **next.gif** and **prev.gif** are good choices for my pages (and since I wrote **relink**, these are the defaults). You can override these defaults in the **links**, if your links look significantly different. If there are no unique patterns identifying your links, you can add an HTML comment to the link line and use that as a pattern.

The LINKS File

We have seen a few simple examples of a **links** file in the discussion above. In addition to page order, you can also specify user-defined link patterns using the following line:

link: *linkname pattern*

The *linkname* identifies the type of link (next, prev, index, or anything you can think of). The *pattern* is a string of characters that must appear on every link of that type. You may override the *next*, *prev*, *toc* (table of contents) and *up* links that **relink** normally works with, and you may define your own links here.

A table of contents file may be identified using the line:

tocfile: *tocname*

Links identified with the *toc* link pattern will generate a link to this file. Unfortunately, **relink** will not update the table of contents with new page orders, so you have to edit the table of contents manually to keep it up to date. Perhaps a future version of **relink** can address this problem.

Nested pages can be specified by using a **{** on a line by itself to start a nested list and a **}** to end a nested list. The page immediately preceding the nested list is called the parent page. The first and last page of a nested list point to the

parent page in there *prev* and *next* links. In addition, each nested page will have an *up* link to the parent page. The *next* link of the parent page will skip over the nested list to the following page. (We assume that the parent page has explicit links into the nested list.)

And finally, separate lists of HTML files can be specified by using a line of dashes. *next*/*prev* links will not cross a line of dashes.

Summing Up

I have found **relink** to be a very useful script in dealing with web documentation, making it very easy to update pages in long documents without worrying about the details of manually adjusting the page links.

Jim Weirich is a software consultant for Compuware specializing in Unix and C++. When he is not working on his web pages, you can find him playing guitar, playing with his kids, or playing with Linux. Comments are welcome at jweirich@one.net or visit him at http://w3.one.net/~jweirich/.

Archive Index Issue Table of Contents

Advanced search

Missing CGI.pm and Other Mysteries

Reuven M. Lerner

Issue #37, May 1997

CGI.pm, for all of its useful and amazing features, is just one of the many terrific Perl 5 modules that isn't included with the standard Perl distribution.

I have reached the point in my career as a columnist when there is too much mail to ignore. I'm not drowning in torrents of e-mail, mind you, but it's still a nice surprise to receive responses from readers. Some of the mail that I have received over the last month or two, though, warrants extended response. In addition to answering some brief questions about CGI.pm and the guestbook program that appeared in the January issue, I will discuss security issues relevant to CGI programmers and Web administrators.

Where Is CGI.pm?

One of the first questions that I received—from several readers of my column in the January issue—is, “Where is CGI.pm?” These readers were surprised that the programs included with my column and which were supposed to work, were failing on them. In particular, they were getting messages like this:

```
Can't locate CGI.pm in @INC at - line 1.
BEGIN failed—compilation aborted at - line 1.
```

What was going wrong here? Why wasn't CGI.pm on their systems?

The simple answer to this question is that CGI.pm, for all of its useful and amazing features, is just one of the many terrific Perl 5 modules that isn't included with the standard Perl distribution. Perl comes with a number of basic modules, but these are only the tip of the iceberg. Most of the modules you might want to use are available from CPAN, the Comprehensive Perl Archive Network—such as those for database server access (obviating the need for separate Perl executables, such as oraperl and sybperl), manipulation of times and dates, handling of e-mail folders, and many more.

CPAN is a set of FTP sites that mirror each other at regular intervals, and which allow programmers to download the most recent versions of various modules programmers have generously donated to the Perl community. One of these modules, and one which I tend to use often in my professional life and in my *LJ* column, is CGI.pm—which, as you might have guessed, is a module that makes it relatively easy for us to write CGI programs.

The easiest way to get to CPAN is via the reflector at perl.com, a site run and maintained by Tom Christiansen, one of the luminaries of the Perl community. If you go to http://www.perl.com/CPAN, making sure you leave off the final slash, you will be able to select a site near you from which you can download various Perl modules. Alternatively, you can include the final slash, as well as the rest of the path name relative to CPAN, and enter a random site from the full CPAN network, as follows:

```
http://www.perl.com/CPAN/modules/by-module/
```

This URL will result in a listing of the various module categories available for downloading. Each category contains one or more modules; for CGI.pm, we need to enter the CGI category, where we can find (as of this writing) the file CGI.pm-2.30.tar.gz.

After downloading this file, use the gunzip program to uncompress the file, and then the tar program to expand it. Do this with these commands:

```
gunzip --verbose CGI.pm-2.30.tar.gz
tar -xvvf CGI.pm-2.30.tar.gz
```

The doubled **v** option specifies additional “verbosity” when expanding files; while you can certainly untar CGI.pm without using any verbosity, I prefer to see what I'm expanding, rather than simply let the command go about its business.

If you are using a system with GNU tar (as is the case with virtually all Linux systems), you can combine these two operations by using the **z** option with tar:

```
tar -zxvvf CGI.pm-2.30.tar.gz
```

After unpacking CGI.pm in this way, you should be able to enter the newly-created directory (named **CGI.pm-2.30** in the above example), configure, and compile the file with the standard Perl module installation commands. Here is what that process looked like on my system:

```
[1008] /downloads% cd CGI.pm-2.30
[1009] /downloads/CGI.pm-2.30% perl Makefile.PL
Checking if your kit is complete...
Looks good
Writing Makefile for CGI
```

```
[1010] /downloads/CGI.pm-2.30% make
cp CGI/Carp.pm ./blib/lib/CGI/Carp.pm
cp CGI/Fast.pm ./blib/lib/CGI/Fast.pm
cp CGI/Push.pm ./blib/lib/CGI/Push.pm
cp CGI.pm ./blib/lib/CGI.pm
Magnifying ./blib/man3/CGI::Fast.3
Magnifying ./blib/man3/CGI::Carp.3
Magnifying ./blib/man3/CGI::Push.3
Magnifying ./blib/man3/CGI.3
```

Now that you have configured and compiled CGI.pm, install it into your system with the command **make install**. In order to do this, you will need to be logged in as the root user, as shown here:

```
[1001] /downloads/CGI.pm-2.30# make install
Skipping /usr/lib/perl5/site_perl/./CGI/Carp.pm (unchanged)
Skipping /usr/lib/perl5/site_perl/./CGI/Fast.pm (unchanged)
Skipping /usr/lib/perl5/site_perl/./CGI/Push.pm (unchanged)
Installing /usr/lib/perl5/site_perl/./CGI.pm
Skipping /usr/lib/perl5/man/man3/./CGI::Fast.3 (unchanged)
Skipping /usr/lib/perl5/man/man3/./CGI::Carp.3 (unchanged)
Skipping /usr/lib/perl5/man/man3/./CGI::Push.3 (unchanged)
Installing /usr/lib/perl5/man/man3/./CGI.3
Writing /usr/lib/perl5/site_perl/i586-linux/auto/CGI/.packlist
Appending installation info to /usr/lib/perl5/i386-linux/5.003/perllocal.pod
```

That's it. Now, **@INC** (the Perl variable that knows where to look for Perl modules) will include CGI.pm, and you will no longer get those nasty error messages complaining that Perl cannot find the file.

Note that Red Hat Linux users might want to use the RPM (Red Hat Package Manager) version of CGI.pm (and other Perl modules) rather than the standard distribution. The advantage of doing this is that installation updates the RPM database and keeps track of the files on your system in an elegant way. The disadvantage is that it often takes a few days or weeks for the latest and greatest version of CGI.pm to appear on the Red Hat servers—and other, less popular modules are sometimes completely unavailable as RPMs. You can find various RPMs at the Red Hat site (and its mirrors), at ftp.redhat.com.

Guestbook Problems

I also received several notes from readers alerting us to two mistakes in the guestbook program in the January issue. Guestbooks, as we all know, generally contain more than one greeting from a user on a site. Thus, if we open the file using the Perl command:

```
open (FILE, ">$filename") || &error_opening_file($filename);
```

we are asking for trouble, since the single **>** operator not only opens the file for writing, but destroys any information the file might have contained previously. The code should really have read:

```
open (FILE, ">>$filename") || &error_opening_file($filename);
```

which means that we want to open the file named in **$filename** for writing, appending our new data to whatever might have been there before. Note that the **>>** operator creates a file if none existed before, so you should feel free to use **>>** for file creation and appending.

The other problem in that program, which was noticed by reader Bill Holloway, had to do with this section of code:

```
@names = $query->param;
# Iterate through each element from the form,
# writing each element to $filename. Separate
# elements with $separation_character defined
# above.
foreach $index (0 .. $#fields)
{
   # Get the input from the appropriate HTML
   # form element
   $input = $query->param($fields[$index]);
   # Remove any instances of
   # $separation_character
   $input =~ s/$separation_character//g;
   # Now add the input to the file
   print FILE $input;
   # Don't print the separation character after
   # the final element
   print FILE $separation_character
      if ($index
}
```

Of course, since we have imported the HTML form elements into the **@names** array, we have to read them out of **@names**, and not out of **@fields**, which is what the above code does. Thus the line:

```
$input = $query->param($fields[$index]);
```

should be replaced with:

```
$input = $query->param($names[$index]);
```

as you can see in the corrected version of the program, which appears in Listing 1.

Individual Users and CGI Directories

Another reader, Maro Shim (writing from Korea), noticed something concerning what I said in the February issue about having to add a **ScriptAlias** or **Exec** directive to the HTTP server's configuration file each time a new CGI directory needed to be added. Maro points out that this means an administrator has to modify the files for each individual user.

Let's get into the pros and cons of letting individual users have their own CGI directories, using Apache as an example. Then we'll discuss why this might not be the best thing to do. Finally, we'll discuss giving each user CGI access, but not giving them the run of the system.

Maro's suggestion is that administrators can create a symbolic link inside the **cgi-bin** directory (which is **/home/httpd/cgi-bin** by default for the copy of Apache running on my Red Hat Linux box), and that this link can point to a directory inside each user's **public_html** directory, which typically contains the user's HTML files.

For example, here is a listing of my personal home directory at the time of this writing:

```
[1068] ~% ls  -F
800omni.pdf    News/          public_html/
Consulting/    Text/          response1.txt
Development/   cgicyrcode.pl  test.dgs
Mail/          chap4de.doc
```

Because I have used the **-F** option to ls, directory names end with slashes, which makes them easier to identify. You can also identify directories by color or boldface text if you use the **--color** option, but I'm too old-fashioned for that. The **public_html** directory is where my personal HTML files reside, which are available via a URL ending with **~reuven/**, since my username is **reuven**, and the web server is configured to look in a user's ppublic_html directory. Thus, if there were a file **index.html**, it would be accessible via the URL:

```
http://localhost/~reuven/index.html
```

(substituting an appropriate hostname for **localhost**, of course).

Personal HTML files are nice, and greatly reduce the amount of work that a system administrator must do in order to run a web server on which dozens, or perhaps hundreds, of users might want to put their own home pages. But what about CGI programs? That's where Maro's letter comes in: Inside the **public_html** directory we can create a subdirectory named **cgi-bin**, as follows:

```
[1071] ~% cd public_html/
[1072] ~/public_html% mkdir cgi-bin
[1073] ~/public_html% ls -F
cgi-bin/   test.html
```

Now the personal HTML directory contains two items—a file, **test.html**, which (in this case) can access **~reuven/test.html**, and a directory named **cgi-bin**, the contents of which I can access as **~reuven/cgi-bin/**. Remember, there isn't any magic in the name **cgi-bin**—at this point, it acts just like any other subdirectory. Indeed, if I were to place the CGI program **elephant.pl** inside **~reuven/public_html/cgi-bin**, I could access it by going to:

```
http://localhost/~reuven/cgi-bin/elephant.pl
```

But rather than seeing the results of executing **elephant.pl**, we will see its source code. This is true because we haven't told our server that it should

execute the program; we need to explicitly install **~reuven/cgi-bin** as a CGI directory. This is the most common way to create personal CGI directories. By including (under Apache) a **ScriptAlias** directive in the file **srm.conf**, we can create new CGI directories for each user on a system. Thus, if we were interested in turning **~reuven/cgi-bin** into a CGI directory, we could use the line:

```
ScriptAlias /~reuven/cgi-bin/ \
/home/reuven/public_html/cgi-bin
```

which would have the desired effect. However, this means that every time we wish to give a user a CGI directory, we need to modify **srm.conf** and restart our HTTP server.

Maro's alternative saves us this work by taking a different approach: Rather than add new **ScriptAlias** directives to **srm.conf**, we simply tell our HTTP server that it should follow symbolic links within the CGI directories that already exist, using the commands:

```
<Directory /home/httpd/cgi-bin>
AllowOverride None
Options FollowSymlinks
</Directory>
```

Once we have done that, we can create symlinks to any directories that we want to turn into CGI directories. For example, to turn **/home/reuven/public_html/cgi-bin/** into a CGI directory, we (as root, or another user with appropriate permissions) would only have to create the symbolic link:

```
ln -s /home/httpd/cgi-bin/reuven \
/home/reuven/public_html/cgi-bin
```

which would then let us use:

```
http://localhost/cgi-bin/reuven/elephant.pl
```

which physically exists in my own personal directory, but which logically exists (as far as the HTTP server is concerned) in the **/cgi-bin** directory, which forces the server to execute it.

Before you turn on CGI directories for individual users, consider the ramifications: CGI programs are potentially an opening from the outside world into your server. If even one CGI program is written with malice aforethought, an attacker could gain access to your system—gathering information about your users, for example, or using that information to alter or damage files. It might seem convenient to give all users access to CGI programs, and it will certainly save you time in the short run, but the security implications are too serious to ignore.

If you cannot restrict CGI to a small subset of the users on your system, then you should consider installing a CGI wrapper program that performs safety

checks before executing these programs. A CGI wrapper is a program which takes a CGI program as its argument. After the wrapper performs several security checks, it executes the CGI program—under the owner's ID, rather than the ID normally reserved for web programs. This prevents one CGI program from reading or changing another program's data—an increasingly possible problem as large numbers of unrelated sites are hosted on the same system.

One such wrapper, known as suEXEC, comes with Apache 1.2. Configuration and compilation of this program is relatively easy and is described in detail in the Apache documentation. Simply put, you compile suEXEC and set it to be SUID root, so it can change to the user ID of the user, regardless of who that owner might be. Finally, you will have to install the **suexec** program outside the normal CGI directory in a location defined in the **httpd.h** file in the Apache source code.

Another popular CGI wrapper is CGIwrap, which works in a similar way without being tied to a particular HTTP server. You can read more about CGIwrap at:

```
http://www.umr.edu/~cgiwrap/
```

It is a good idea for these wrappers to run CGI programs under a user ID other than your HTTP server's default, letting individual users write and install various programs of their choosing, the possibility of sending programs data that can overflow buffers, or that might pass malicious arguments to programs using the Unix shell is too great to ignore, particularly with the security holes for which Unix is famous. You might want to insist that any CGI program on your server written in Perl use the **-T** argument, which turns on Perl's **taint** system that prevents user data from being passed to the shell without going through some sort of filter—but of course, such checks can be ignored, and not all CGI programs are written in Perl.

In short, there isn't any perfect solution, which means that at some point you will have to decide whether to make your system safer (but with angry users), or more exposed to possible damage (but with users satisfied with their ability to run CGI programs of their choosing).

Permissions for CGI Programs

While we're on the subject of security, this is probably a good time for me to publicly wipe away some of the egg that remains on my face in the wake of my February column, in which I suggested that you should install CGI programs with permissions of 777, known to non-numeric types as “a+rwx”, or permission for all users on the system to read, write, and execute the program.

Suffice it to say that this is a grave error, as several readers noticed. Computer security depends on plugging as many holes as possible. On networked multiuser systems running programs that come from various sources, it's almost certainly a bad idea to install a program having permissions that let anyone on the system modify the contents of that program, particularly when a simple (and probably hard-to-notice) modification or two can turn a seemingly innocuous program into a ravenous bug-blatter beast. On a system not running one of the wrappers mentioned here, all CGI programs are run with the same permission, meaning that someone could write a program that can mess with the code or data of another.

If you are the only programmer working on a particular CGI program or Web site, then you can install your programs with 755 permission (u=rwx,ga+rx), so that others on the system—including the HTTP server, which is generally responsible for running CGI programs—can read and execute your code but cannot modify it.

If you are working with others on a site or CGI program, you can set the permissions to 775 (ug=rwx,a+rx), which lets everyone read and execute the program, but allows only the owner and members of the file's group to edit it.

There are probably times when it is appropriate to install a CGI program with 777 (a+rwx) permission, but these are rare.

That's it for the mailbag for this time. Next month, we'll return to a discussion of how to make life easier for non-programmers who might want to modify entries in tables on disk, by writing a few small CGI programs which can read and write files efficiently and easily.

Reuven M. Lerner has been playing with the Web since early 1993, when it seemed like more like a fun toy than the World's Next Great Medium. He currently works as a independent Internet and Web consultant from his apartment in Haifa, Israel. When not working on the Web or volunteering in informal educational programs, he enjoys reading on just about any subject, but particularly politics and philosophy, cooking, solving crossword puzzles and hiking. You can reach him at reuven@the-tech.mit.edu or reuven@netvision.net.il.

Archive Index Issue Table of Contents

Advanced search

World Wide Web Journal

Danny Yee

Issue #37, May 1997

The range of topics covered is immense.

- ISBN: 1-56592-211-5
- Publisher: O'Reilly & Associates
- US$24.95 per issue, US$75.00 per year
- Reviewer: Danny Yee

Issue 1 of the *World Wide Web Journal* contained fifty-nine papers, fifty-seven from the Fourth International World Wide Web Conference (held in Boston in December 1995) and two from regional conferences. The range of topics covered is immense. To list just a few (in no particular order): why the GIF and JPEG formats aren't good enough for really high quality graphics; low-level security in Java; the results from the 3rd WWW Survey; an analysis of Metacrawler use; caching systems; a filtering system to provide restricted access to the Web; a PGP/CCI system for Web security; the Millicent system for financial transactions involving small sums; smart tokens; and better support for real-time video and audio. There are also papers on the use of the Web in education, on cooperative authoring tools, on Web interfaces to database and software systems, and a cornucopia of other things.

Issue 2 was a disappointment. It consisted solely of standards documents: Requests For Comment (RFCs) numbers 1630 (URIs), 1808 (Relative URLs), 1736 (IRL recommendations), 1866 (HTML 2.0), 1867 (Form-Based Upload), and unallocated (HTML Tables); Internet drafts on HTTP 1.0, PEP HTTP/1.1, and HTML Internationalization; and W3C drafts on PNG and Cascading Style Sheets. Since all of these documents are freely and easily available on-line and several have already been superseded, this is really of limited value. (Nicely formatted bound versions of standards documents are useful, but only for the standards that have some sort of permanence.)

Though shorter, issues 3 and 4 strike a better balance between background material, standards and technical papers. As background material, issue three contains an interview with Tim Berners-Lee and descriptions of other World Wide Web Consortium staff. The technical papers are mostly about Web demographics and "geography": the Nielsen/CommerceNet, GVU, and White House surveys; systems for statistical analysis of traffic; visualisation of Web connectivity and traffic; and the implementation of national Web cache systems in the United Kingdom and New Zealand. Issue 4 is mostly devoted to HTTP: it contains technical specifications for and informal descriptions of HTTP 1.1, as well as papers on state management (cookies), digest authentication, and future directions for HTTP. There are also papers on PICS, PNG, distributed objects, and distributed authoring.

Though few assume much technical background, the papers in *World Wide Web Journal* are mostly technical in focus: they are not for everyone who runs a Web server or authors HTML. However, for those concerned with the future of Web technology—because they are directly involved in protocol or system development, because they need to prepare for future applications or out of simple curiosity—the journal is a good way of keeping up with the most important developments. As a quarterly journal, it fills a niche between books and information sources on the Web itself.

World Wide Web Journal can be sampled on the Web at http://www.w3.org/pub/WWW/Journal/.

Danny Yee receives a complimentary subscription to World Wide Web Journal but has no stake—financial or otherwise—in its success. He can be reached at danny@cs.su.oz.au.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #37, May 1997

Readers sound off.

Code Examples

Any chance you could start putting the *LJ* code examples on the web site, so I don't have to type the stuff in? Especially hairy stuff like the PGP patch on page 6 of the February '97 issue. Sheesh. Really, I think this would be great. —Peter Watkins Washington, DC peterw@clark.net

We think this is a good idea too, and starting with this issue, example code can be found at ftp://ftp.linuxjournal.com/lj/listings/issue##/. The files are tar, gzip files, one for each article, named article##.tar.gz. There will be a footnote to each article that has listings in it, giving you the article number.

Acronym Use

My first question is, “What does CGI stand for?” The second question is, “Why did the editor/s never ensure that the abbreviation was defined at least once in the magazine?” This lack of definition of acronyms is very annoying to me since the computer world is so chock full of acronyms. Acronyms in this environment are also very context-sensitive—so much so that defining terms like this should be mandatory in *every* article published anywhere. —Mac Bowles Senior Software Engineer Lockheed Martin Astronautics kbowles@claven.mmc.den.com

First, CGI stands for Common Gateway Interface. Second, I agree that tossing acronyms around without defining them is annoying, and plan to be more careful in the future.

Linux in Schools

I have read the letter to *Linux Journal* from Mr. Jack McGregor, who is pushing usage of dumb terminals for schools (low cost). I am a Linux consultant. However, I am not pushing dumb terminals, but Linux-based X terminals using old 386 and 486 PCs. My experience with low end 486s demonstrates that indeed they are very fast and fun to use to operate major desktop packages (WordPerfect, Netscape, Applixware and StarOffice are a few I have tried). They can also run games (e.g., Doom).

I have customers who have turned to this solution not to save money but to gain raw speed. For example, one is using a dual Pentium Pro 200MHz with 192MB of RAM as a server for 10 users (C++ developers). Everything is in RAM all the time for all users. This beats any network when it comes to loading software or searching through directories and so on. In one case, the speedup they are getting by sharing the same server instead of the more standard Windows-to-server relation is close to a factor of 10.

While X is an old technology for some, Linux is making it into a revolution because of its low cost. —Jacques Gelinas jacques@solucorp.qc.ca http://www.solucorp.qc.ca/linuxconf/

Red Hat Install Problems

I have been using the Slackware distribution 3.0 (1.2.13 kernel) for over a year. I wanted to upgrade to the 2.0 kernel, and decided that a new CD-ROM distribution would be convenient. After reading about Red Hat 4.0 in the *LJ* Readers' Choice Awards and an *InfoWorld* review that selected Red Hat 4.0 as one of the two best operating system releases in 1996 (the other was NT workstation), I decided to order.

My installations (about a dozen trials) were plagued with random segmentation faults, stack-dumps and reboots. The Red Hat support team responded as advertised and suggested hardware (i.e., CPU, RAM, cache, motherboard) was producing my problems. The Red Hat technician pointed me toward RAM or cache because I just had a brand-new motherboard installed, and he presumed that the motherboard or the brand-new cache was not a problem. Finally, the Red Hat technician suggested that:

“Red Hat is sometimes not able to run (for unknown reasons) on some hardware that *will* run Slackware.” (E-mail from Red Hat support.)

I had my RAM diagnosed by a local computer repair shop that has a hardware technician who is a also Linux guru. No problems were reported with the RAM, but the technician could not duplicate my installation symptoms.

Finally, a bit dazed and still suspecting the RAM, I purchased some extra RAM. I tried the installation one last time, using only the new RAM—I still could not successfully install Red Hat 4.0. Alas, I am back to the Slackware 3.0 and out $60 for the Red Hat.

I am truly disappointed that I cannot get Red Hat 4.0 working. It seems Red Hat has so much to offer new Linux users in terms of configuration, installation, etc. But, as Microsoft can attest, it will be difficult for any commercial distribution to support every PC configuration. My PC is the evidence.

All is not lost, though. As the web master for our software manufacturing firm, I take care of the Intranet Web pages. We need a new internal web server, and I am adamant that it runs on a Linux box. Maybe the Red Hat distribution can foot this bill. For my PC though, I am sticking with Slackware. —Jeffery C. Cann Software Engineer jcann@intersw.com

Shopping for Linux

Cory Plock wonders why he can't find Linux distributions on the shelves of local software stores (*LJ* February 1997). Perhaps he's just living in a technologically repressed area. I just checked my nearest shopping-mall computer store here in Quebec City: They have the 6-CD InfoMagic package and the 4-CD Walnut Creek distribution, both up to date and competitively priced. A nearby store has several copies of two books on Linux. The distribution mechanisms must exist —encourage local dealers to use them. —Don Galbraith dsg@clic.net

On-line Linux Users Group

Hi. I have been a long time reader of *LJ* and it has been a great help to me, and I am sure that applies to many in the Linux community. Now, my friends on the Net and I have also done something as a contribution to Linux which I thought would be interesting to you and helpful to your readers. This is to create an On-Line Linux Users Group for people interested in learning more about Linux, providing help to other Linuxers and promoting Linux. Our web address is: http://www.linuxware.com/. —Peter Lazecky peteri@linuxware.com

AutoMount Article Comment

I am using the version of amd that comes with Red Hat 4.0. The NFS hosts are running SunOS 4.1.4 and Solaris. I found the suggestion that amd is relatively bug free incorrect. In the first few days of using it I found two important bugs. The first is that it confused the node name of one machine with the IP address of another machine. That is, I found directories from one machine under the name of the other in the /net directory. The second bug I have experienced several times. If a directory is unmounted, amd doesn't seem to know how to

mount it for several minutes afterwards. Both of these bugs result in directories disappearing—including my home directory. It can make it very hard to justify using Linux when these major type problems exist.

As you can see, this is a big issue for me. I have seen several postings in dejanews referring to other problems with amd. I would like to see more support, but up to this point I have found very little in the way of answers to most questions about amd. Thanks for the article anyway. —David Uhrenholdt duhrenho@vette.sanders.com

Larry Wall Article Comment

I read Larry Wall's article on Perl on the way in to work today. My work involves C, Korn shell and Perl. I am convinced that Perl is a marvelous language. Mr. Wall's article supports that.

I understand the notion of creativity as a function of a large palette (...there's more than one way....) and the theory that “form follows function”. My conclusion is that Perl is unacceptable as a development tool because I cannot support it. It takes too long to discover (glean, figure out, guess at, puzzle out) which of the myriad possible methods was used by the original developer—even when that was me. I will continue to write Perl for fun and use more documentable, supportable languages for important systems.

I also wonder if Mr. Wall's writing would be a little more effective if he didn't attempt to be funny in every paragraph. He is the linguistic equivalent of the aggressive graphics that prevent many people from being able to read that very trendy San Francisco- based magazine on pop-wired culture. —Brandon Sussman #VATAcc70713@vantage.fmr.com

Another Algorithm for Polygons

5 Feb 1997: I enjoyed the interesting article by Bob Stein on algorithms for deciding whether a point is in a polygon in your March issue (“A Point about Polygons”). It is too bad that Bob wasn't familiar with the algorithm used for this in the WN web server (see http://hopf.math.nwu.edu/), as it has some interesting similarities and differences when compared to the algorithm he describes. Like Stein's it uses integer (actually long int) arithmetic rather than floating point.

I first used this algorithm in a version of WN released in July of 1995. As with Stein's algorithm we start by translating, so the test point is at the origin. Instead of counting the parity (evenness or oddness) of the number of crossings with the positive Y-axis, the actual signed number of crossings is counted (I used the positive X-axis instead of the Y-axis, but that is immaterial).

More precisely, the algorithm counts +2 if an edge crosses the positive X-axis with positive slope and -2 if it crosses with negative slope. If an edge *ends* on the positive X-axis, it gets a count of only +1 or -1 depending on the slope. If the edge lies entirely in the positive X-axis, it gets a count of 0. If the origin (test point) is actually on any edge, we declare that the point is in the polygon and quit. After all edges have been counted, we declare that the test point is outside the polygon only if the total count is zero.

The implementation in WN is about three times as long as Stein's implementation, largely because I wanted to get all the special cases right even if in practice they don't matter much. In particular, if the test point is on an edge, it is always declared in the polygon. Also polygons with only two sides or degenerate polygons (like points) work properly.

There is one very big difference in the way the two algorithms behave when the polygon is not simple (i.e. crosses itself). Imagine a five-pointed star drawn in the usual way without lifting your pencil from the paper. With the even/odd count only points in the five triangular “tips” of the star will be considered inside while points in the pentagonal central region will be considered outside. The WN algorithm, on the other hand, will count all these points, tips and center, as “inside”. This is, in fact, the reason I chose this method rather than the even/odd count.

The reason this difference occurs is not too difficult to understand. Imagine the polygon is a stretched rubber band held in place on a table with thumb tacks at each vertex. At the test point we erect a vertical post perpendicular to the table. Now remove all the tacks and let the rubber band contract into the post. It may wrap around the post some number of times positively or negatively (i.e., counter-clockwise or clockwise) or it may not be hooked on the post at all. The even/odd algorithm is counting whether the number of times it wraps around is even or odd, while the WN algorithm is counting the full number.

If the polygon does not cross itself the actual number can only be 0, +1, or -1, so the even/odd algorithm works fine. With the five-pointed star, if the post is put in the central region, the rubber band goes around twice, and so the algorithms give different answers.

If anyone is interested in the WN implementation, just download the distribution and in the file wn/image.c look for the functions **dopoly()** and **segment()**. The distribution can be found at http://hopf.math.nwu.edu/. —John Franks john@math.nwu.edu

Archive Index Issue Table of Contents

Advanced search

Changes at *LJ*

Marjorie Richardson

Issue #37, May 1997

Gary is now SSC's Technical Editor and I am Managing Editor of *Linux Journal*

In an amazing act of prestidigitation Phil Hughes waved his hands and Gary Moore and I traded positions on February 1. As a result, Gary is now SSC's Technical Editor and I am Managing Editor of *Linux Journal*. This is one of those win-win situations—Gary gets to spend more time editing and I get to tell everyone what to do. I love managing things. I suppose I have a strong controller streak to my psyche.

I have worked with computers for over 15 years now, mainly programming scientific applications in Fortran for geophysical (oil) companies. I also did a lot of technical writing in the form of software documentation. The writing was always the most fun, so I've enjoyed my work here at SSC and expect to continue to do so as Editor of *Linux Journal*. Since coming to work for SSC, I've done editing of reference cards, such as Java and HTML, as well as lots of copy editing for *Linux Journal*. And of course, I made sure that *Linux Gazette* got posted every month. Actually, it was the time spent on *LG* that convinced me that I could handle the job for *LJ*. I have retained custody of *Linux Gazette*. I have too much fun with *LG* to give it up, and I intend to have just as much fun with *Linux Journal*.

A Couple of News Items

The project led by Alan Cox for Linux users to sponsor a penguin at Bristol Zoo in Swansea, UK is now complete. The sponsorship was done in Linus Torvalds's name as a 1996 Christmas present. Details can be found at http://penguin.uk.linux.org. Sounds like a fun project for user groups.

The first virus able to infect a Linux system has been found by McAfee Associates. The virus, named Bliss, has spread to Linux systems, as many Linux

users play Internet games while logged in as root. To learn how to avoid this danger, check out this month's article “Safely Running Programs as Root”, by Phil Hughes.

If you have a spare Linux CD to give away, you can list your e-mail address at http://emile.math.ucsb.edu:8000/giveaway.html. Those people who need them will contact you, send you a self-addressed stamped envelope and then you can send them the CD. If you would prefer to lend a Linux CD locally, you can also sign up to do that at the site. This is a worthy project that should help to spread the word about Linux.

This Month

This month our focus is on Linux ports, and we have several articles on different ports including Alpha, Mac and the PowerPC. Also, thanks to Alessandro Rubini, *Kernel Korner* has returned to our pages.

Beginning with this issue *Linux Journal* will have tar, gzip files containing the listings for our articles. You can grab the files for this issue from ftp://ftp.linuxjournal.com/lj/listings/issue37. As time permits, we will add files containing article listings from previous months.

Next Month

Next month we'll focus on networking. Planned are articles on multi-platform networking with Linux, on communicating between home and office and on setting up a Sun SPARCstation.

Send me any ideas or suggestions you might have for articles or *Linux Journal* in general. My e-mail address is info@linuxjournal.com. —*Margie Richardson*
Managing Editor

Archive Index Issue Table of Contents

Advanced search

Linux and Web Browsers

Phil Hughes

Issue #37, May 1997

Netscape 3.01 for Linux exists and is still not supported.

Back in *LJ* issue 20, I wrote about how Netscape said they intended to drop Linux from the list of supported operating systems. To most of us this wouldn't have been a surprise as Linux has never been supported by Netscape—only an unsupported version of Netscape has been available for Linux.

Well, things haven't changed much. Netscape 3.01 for Linux exists and is still not supported. In that same article I suggested that if we could write a complete operating system as a community effort that we could do the same for a web browser. Then I went on to suggest that starting with Arena, the W3C's test platform, and building the best web browser for Linux from it was a reasonable idea.

It probably was a reasonable idea, but it never seriously happened. We all continue to use Netscape or Mosaic and hope for the best. Another thing happened recently that makes me nervous: the Mosaic 2.8 team was moved to another project, so we really are pretty much at the Netscape or nothing stage.

The Lights Come On

I was thinking about this yesterday while reading the Linux newsgroups looking for a possible topic for this column. The answer was there. There was a press release from Yggdrasil Computing that announced that they would be working on development of Arena. To quote the announcement,

> The World Wide Web Consortium has approved Yggdrasil Computing to coordinate future development of Arena, a powerful graphical web browser originally developed as the Consortium's research test bed.

All the work will be under the GPL, meaning that it will be available to anyone—commercial or non-commercial. This isn't a Linux-only effort. Yggdrasil also plans to make it available on other Unix platforms and MS-Windows. The MS-Windows version will be accomplished by joining forces with Pearl Software which offers an X-Windows emulator.

More Players

I suggested this topic to Margie Richardson, *LJ*'s Managing Editor and also the Editor of *Linux Gazette*, our on-line Linux magazine (http://www.ssc.com/lg/), and she handed me information on another effort called the *Linux Browser Project*. I went off web searching and found that there is another alternative to Netscape in the making.

The first thing I found was that the project has been renamed to *Mnemonic*. This is because, while Linux is the development platform of choice, the goal is to produce a free browser available for many different operating systems. To start, here is the "What is" from their web page:

> The basic goal of Mnemonic is to produce a free, usable and maintained World Wide Web Browser for many different operating systems. The intent is to make the browser as modular as possible, to make it easy to add new features and to port to different interfaces and platforms. The base browser will most likely support HTML 3.2 and Cascading Style Sheets, with support for things like Java and HTML Extensions being distributed as add-on modules. Other proposed features include IPv6 support, the ability to auto-download modules when needed, and a highly customizable user interface.

Sounds good so far. But, why another project? Well, they have a page that addresses that on their web site. They suggest that configurability and a modular architecture is what has been missing from other browsers. This was certainly true of Mosaic where a virtual re-write was started.

This modular approach includes the user interface. That means that those who love Motif will be able to use a Motif UI, those who love Tk will be able to use a Tk UI and so on. They also have a projected release date of July 14, 1997, which makes you think that they are serious.

Both of these projects are for free software. And Linux has proved that developing in a free environment can produce viable products. In fact, the Arena project predated Netscape Navigator and Microsoft Internet Explorer and some innovations in Arena were later used in these commercial products. If you have interest in the Web and are looking for a project, check these out.

Where to Find Out More

Linux Trademark Status

Archive Index Issue Table of Contents

Advanced search

Connecting SSC via Wireless Modem

Liem Bahnemann

Issue #37, May 1997

Having fun using wireless modems as a LAN bridge between SSC offices.

Recently, SSC needed more room and rented another office. The remote office needed a connectivity solution to link it to the original office's LAN. Several options were available to us:

1. Frame relay (56k, T1...)Advantage: high speedDisadvantage: high cost of line and hardware
2. ISDNAdvantage: relatively high speedDisadvantage: potentially expensive hardware, long wait for installation
3. 28.8k serialAdvantage: cheap hardware, readily availableDisadvantage: low speed
4. Wireless networkingsimilar to ISDN
5. Guerrilla Ethernet (run our own coax down the street) Advantage: 1.555-> 2.5 Mbits fastDisadvantage: street sweepers

We explored each possibility, and decided that outside of the probably illegal guerrilla Ethernet, wireless modems would be the most fun to install and use. We contacted various vendors for wireless products and found Freewave had the highest-speed wireless modems available. Freewave sent us a pair of modems to test at the office.

The Freewave modems we received were small and easy to configure. The manual stated they could potentially be used at a range of 20 miles, line-of-sight. We would soon see if that was a true statement.

How Wireless Modems Work

Freewave wireless modems act as a null-modem cable, and unlike regular modems, do not need dialing. The modems *look* for other modems on their

frequency and link to them. Wireless modems can link point-to-point, i.e., two modems, or multiple modems can be connected to a single modem that acts as a hub. Point-to-point, the modems are configured as a master and a slave. One calls the other and a link is established. The *number* they call is an internal, firmware encoded number.

Configuration

The firmware of the Freewave modem is accessed by pressing a dimple on the front plate of the unit, which puts the modem into configure mode. Connecting to the serial device with **minicom** or **xc** at 38400 bps enables you to access the firmware menu. You then have the options to configure the speed the unit will use to talk with other modems, the numbers of the other Freewaves to call, and the behavior of the unit with the other modems—as a slave or a master.

Since we wanted point-to-point usage, the number was set to the number of the second modem, and the speed was set to 115200 bps.

Getting the Modems Talking

Exiting the firmware puts the unit back in communication mode. It took several configuration tries to get each Freewave configured with the right speed as master and slave. Finally, the status lights on the units showed us a link, and flashes of packets could periodically be seen. We noticed that at 115200 bps, with the modems 20 feet apart, we weren't communicating well. Characters could not be sent either way even though a link was definitely established. Our communication test initially consisted of using minicom to send “Hello? Is it working? See this?” back and forth between modems.

We dropped the speed back to 9600 and established a reliable, clean link. We also discovered that sometimes the firmware on one of the modems would suddenly configure itself to be the master instead of the slave and talk at odd baud rates like 230400 bps.

After an hour or two of playing with the firmware, we were able to get a reliable 115200 bps connection at 20 feet. Now it was time to test a link between the two offices. The new office is south of the old at a distance of approximately 1500 feet with a large nursing home and several houses in between (read: not line-of-sight). After spending 2 or 3 hours beating on bad serial ports and slow 16450 UART-equipped hardware, we finally built a system with 16550As and were able to test the modem. The modem was placed in the window, just in case an extra wall might make a difference. After a bit more banging on the firmware, we finally established a connection with the office at 115200. We had link speed and reliability with a couple of transfers of the Linux kernel back and forth, and we also met with some problems. Occasionally the link would freeze,

which meant resetting the modem, which in turn sometimes caused a glitch in the firmware. Other times it would transfer flawlessly at 7-8Kbytes/sec, a respectable performance. All in all, we decided that this performance, though quite good, was not robust enough to act as a LAN bridge to the remote office. Therefore, we packed up the modems and sent them back to Freewave.

I believe that with line-of-sight or at least minimally-blocked usage these modems could yield quality results. They're the fastest wireless modems we found—others we researched had maximum rates of 9600 bps. The initial cost was high, at about $1500 per unit, but given the costs of similar performance hardware, such as ISDN routers, DSU/CSUs and such, the overall cost was actually low, since there are no line charges. With improved reception (boosters were very costly) these units would have served their purpose quite well. Without it we decided to use an ISDN instead.

Liem Bahneman quit school at the University of Washington to pursue more important research into the area of Linux-induced insomnia. Nowadays, when Liem isn't making sure the web servers at the Cobalt Group aren't crashing, he's likely to be found playing with Legos or his Star Wars memorabilia collection. He can be reached via e-mail at roland@cobaltgroup.com.

Archive Index Issue Table of Contents

Advanced search

Paths

Lynda Williams

Issue #37, May 1997

For all the newbies out there who have just acquired their first computer, here's the map for finding your way around the many paths inside it.

Starting at Home

After logging in, type **pwd** at your Linux prompt, and you will see something like this:

```
/home/williams
```

This is a path; in this case, the path for my home directory. A directory is where files are kept. They can also contain other directories. The path /home/williams is shorthand for “The directory williams which is in the directory home, local, which is in the root directory of the file system.” Each word between / characters is a name; what it names is a directory. The pwd command reported /home/williams to me, because it was my “Present Working Directory,” more commonly known as the current or default directory.

The current directory is a simple idea with many faces. Immediately after logging in, it is your home directory. Later, it could be any directory on your file system. Your current directory is really that of your shell. Other processes have their own, which change independently. A single process with multiple sub-windows may even maintain a current directory for each.

It is useful to be aware that a command-line user will say “I am in /etc/skel”, while a graphical interface user of the same system would report that /etc/skel is “open”. Both mean the same thing. The current directory is the assumed target of your commands and any files requested. Paths let you refer to other directories. Unfortunately, learning to use them can be needlessly painful due to the historical choice of nomenclature.

Puzzling Nomenclature

Nomenclature describes how things are named. The / symbol, in path names, suffers from a nomenclature quirk known as overloading, which means it can be correctly interpreted in more than one way.

If a path begins with /, the meaning is “the root”. The root is itself a directory. [Note that /, meaning the root directory of the file system, should not be confused with /root, a directory in the / directory—Ed] Think of it as the boss directory which isn't contained by any other. Beginners often fail to “see” root at all, because the visual impact of a leading / is small. This is a serious error. A / between two directory names, after root, merely separates them. The / is not a directory—it is just a delimiter. Try this simple test to see if you've got it. How many directories are there in /home/williams?

The answer is three, not two. The two most first-year students in my course get are /home and /home/williams. About a quarter every term fail to count the root itself. If we step beyond paths to files, the nomenclature becomes ambiguous. No matter how closely we inspect /home/williams/foo, we cannot deduce whether foo is a file or a directory. If the name were /home/williams/foo.txt, we might believe foo.txt is a file, but it is possible for directories to have extensions, and common for files to omit them.

What do the following two examples tell you about the names *lists* and *weba*? (The cd [Change Directory] command resets your current directory.)

```
Example 1
$ cd /home/lynda/lists
$ pwd
/home/lynda/lists
```

```
Example 2
$ cd /home/lynda/weba
bash: /home/lynda/weba: Not a directory
$ pwd
/home/ftp/pub/weba
```

You should be able to deduce that *lists* is a directory, but *weba* isn't. Of course, there are plenty of ways, starting with past experience, to resolve ambiguity. The **file** command is another, as are the **-F** and **-l** options for the **ls** command.

Relative paths

So far, all my paths have begun with the root. Path names which begin this way have the same meaning no matter what the current directory is. They are called **absolute** paths. For example, the command:

```
$ cd /home/ftp/pub/weba
```

will make /home/ftp/pub/weba my new current directory regardless of the existing one. What would the command below achieve?

```
$ cd weba
```

Any reference to a file or directory which does not begin with a slash (/) is relative. This begs the question “relative to what?” Relative to the current directory. The command above, therefore, will work as desired only if my current directory is already /home/ftp/pub. Consider the following attempts to use relative paths to change directories. The prompt has changed to show the current directory before the $ symbol, and I have taken liberties with spacing to enhance readability.

```
Example 1 (success)
/home $ cd ftp/pub
/home/ftp/pub $
Example 2 (failure)
/home $ cd pub
bash: pub: No such file or directory
/home $
example 3 (failure)
/home $ cd /ftp/pub
bash: /ftp/pub: No such file or directory
/home $
example 4 (success)
/home $ cd ..
/ $
```

Relative paths are just absolute paths with the current directory assumed as a prefix. They are more common than absolute ones, as we tend to work out of the directory of greatest interest to us at the moment. Note, however, that it is wrong to start one with a /, as in example 3, because doing so makes the path absolute. Sticklers for syntax will note we lose a delimiter / in the bargain.

Two special components in building relative paths, are . and .. which stand for “this directory” and “the directory containing this directory ”, respectively. You can use them alone, as in example 4, or with other components as in this fairly typical example of backing up and going forward to a different directory:

```
/home/ftp $     cd ../williams
/home/williams $
```

Building with Paths

Confidence in using paths helps with a surprising variety of more complex topics. The environment variable $PATH is a list of ordinary paths separated by colons. The shell uses it to search for program files to execute. Many programs use special variables to establish the base path for their operations; it is useful to know how to construct a path when one is called for as a command parameter.

We saw how paths feature in the absolute names of files. They also appear in URLs (Uniform Resource Locators, commonly associated with the World Wide

Web). Paths may be one of the first things you learn, but their applications are endless.

Lynda Williams (http://quarles.unbc.edu/ljw.html) mentors faculty at the University of Northern B.C. in the development of web sites, supported by her Linux server, (http://vaughan.fac.unbc.edu/ctl) and teaches survival tactics for computing to first-year students in her computing applications course (http://quarles.unbc.edu/cpsc150). Her pioneering work in the community networking movement (telnet freenet.unbc.edu) is another expression of her interest in popularizing computing technology. She can be reached via e-mail at williaml@unbc.edu.

Archive Index Issue Table of Contents

Advanced search

ncpfs—Novell Netware Connectivity for Linux

Shay Rojansky

Issue #37, May 1997

Linux supports a very wide array of networking protocol, and software exists to tap into virtually any network server, and even become a server for non-Unix clients.

Today's networks are becoming increasingly complex and diverse. Often a system administrator is forced to face a network of more than one operating systems, and sometimes even more than one communication protocols. Not surprisingly, one platform that manages to adjust in these harsh conditions is Linux. Linux supports a very wide array of networking protocol, and software exists to tap into virtually any network server, and even become a server for non-Unix clients. Samba provides client/server for Windows 3.11/95/NT networks, Netatalk takes care of Macintosh's Appletalk, and in this article I will discuss yet another program that allows any Linux machine to become a full-fledged Novell client.

Novell networks are among the most popular ones in the world. Therefore, it is no surprise that some means of interacting with Novell servers has evolved. A typical Novell network consists of one Novell server, usually running dedicated to Novell, and many clients (PCs usually running DOS/Windows). Unlike NFS Unix networks, there is a very big software difference between the Novell server (usually running a special OS) and the clients (usually running DOS/Windows with Novell drivers). While commercial products exist that enable interaction between Novell and Unix systems, ncpfs provides a powerful, easy and free way of doing it.

ncpfs is a suite of programs created and maintained by Volker Lendecke (lendecke@namu01.Num.Math.Uni-Goettingen.de) that let you access a Novell server in various ways. The primary service a Novell server provides is its files. A Novell server holds several volumes, each usually corresponding to a hard drive or CD-ROM. ncpfs lets you easily mount a Novell server—the directory used to mount the server will contain a directory for each volume accessible on that

server, and in those directories will be the actual directories and files. Note that a Novell server allows you to see only what you have permission to see.

How to Use NCPFS

Get the latest version of ncpfs from: ftp.gwdg.de:/pub/linux/misc/ncpfs or from: sunsite.unc.edu:/pub/Linux/system/Filesystems/ncpfs. Untar it, and compile the tools by typing **make** and **make install**. Consult the README file, if you have any problems.

ncpfs utilizes the NCP (Novell Core Protocol) protocol, which sits on top of IPX (Internetworking Packet eXchange). First, make sure that IPX support is available in the kernel (or as a loadable module). Then, you must configure the IPX interface. ncpfs comes with the Linux IPX tools, which allow you to create an IPX interface and a route, somewhat like ifconfig and route. The easiest way to configure your IPX system is by doing this:

```
ipx_configure --auto_interface=on \
        --auto_primary=on
```

This attempts to automatically determine everything about your interface, and to set it as the primary one. If this doesn't work, you will have to try to configure manually. For more information consult the man pages for ipx_configure, ipx_interface, ipx_internal_net and ipx_route. Now you are ready to run ncpfs utilities.

All the ncpfs tools work in a similar fashion. Since each operation requires accessing a Novell server, almost each command execution requires that three things be supplied: the server name, the user name and the password. There are two ways to do so:

1. Use command-line parameters: **<command> -S <server name> -U <user name> -P <password>**. This is usually a tiresome method since EVERY command needs to have these three switches fed to it.
2. The file ~/.nwclient may contain information about servers. Each line may contain information in the following syntax: **<server_name>/<user_name> <password>**

If you specify the **-S** command-line parameter, the program will automatically get the user name and password from the appropriate line of this file. If not, it will use the first line.

To cut straight to the interesting stuff, in order to mount a Novell server, simply type:

```
ncpmount <mount_point>
```

Again, add switches for the server, user and password or use ~/.nwclient.

Your mount point will contain a directory for each volume, containing the actual files, in the Novell server. **ncpmount** also provides many options to control the mounting, such as the UID and GID of the file hierarchy. Consult ncpmount.8 for more details. Note that a Novell server can be mounted several times from the same computer. Note also that **ncpmount** and **ncpumount** do NOT have to be setuid, which enables any normal user to mount their accounts on a Novell server, opening yet more possibilities for ncpfs application in the real world. For example, to access the file \LOGIN\LOGIN.EXE on volume SYS, on the Novell server MYSERV on /mnt, as the user **supervisor** with the password **12345** (let's hope there aren't many of these out there), execute:

```
ncpmount -S MYSERV -U supervisor -P 12345 /mnt
```

OR have the following line in ~/.nwclient:

```
MYSERV/supervisor 12345
```

and execute:

```
ncpmount /mnt
```

Once the Novell server is mounted, the file LOGIN.EXE will be represented as /mnt/sys/login/login.exe.

In order to print to a Novell server, simply execute:

```
nprint -q <queue_name> <file>
```

This will contact the specified printer queue on the server and send it **<file>** as a print job. See nprint.1 for more details. Note that ncpfs also provides a print server, allowing Linux to connect to a Novell server's queues and transfer jobs to the Linux printing system; see pserver.1 for more information.

Another important functionality provided by ncpfs is direct access to the bindery. The bindery is the database where a Novell server keeps all information about users, groups, and just about everything else. Unfortunately, the bindery can normally be accessed only by using tools provided by Novell. While these tools are usually very colorful and user-friendly, when it comes to manipulating hundreds of users and groups they don't pack the punch. In Unix this problem is solved by providing direct access to the database—/etc/passwd, for example, and using general-purpose tools such as sed, awk and perl. ncpfs provides tools to access the bindery and modify it, allowing the savvy system administrator to write flexible shellscripts to modify a Novell server's bindery. So, for example, if you wish to change every single user's name so that the third letter is x, you can do so quite easily. This ability means that even if you don't

need to access a Novell server from a Linux machine you might still find a use for ncpfs for administrative purposes.

The tools nwbocreate, nwbols, nwboprops and nwborm allow you to manipulate bindery objects (such as users, groups, print queues, etc.); the tools nwbpadd, nwbpcreate, nwbprm, nwbpset and nwbpvalue will change the properties of objects. These base-functionality programs open up endless possibilities for Novell management utilities for Linux, even more diverse than the ones that exist for DOS/Windows, since no programming libraries are normally provided with Novell Netware. See their man pages for additional information.

Some more nifty tools provided by ncpfs are:

- **nwrights**, **nwgrant**, **nwrevoke** allow the modification of file access rights like Unix's chmod)
- **nsend** sends a message to a user via the Novell server (note that if the recipient is also using ncpfs, their computer must run kerneld to receive the message)
- **slist** lists the Novell servers available on the network;
- **nwpasswd** changes the password of a user;
- **pqlis** lists the print queues available on a Novell server;
- **nwuserlist** lists the users logged into the server and their hardware addresses
- **ncopy** copies files within a Novell server without sending them through the network

A Use of ncpfs in Real Life:

My school, the Hebrew University High School in Jerusalem Israel (www.leyada.jlm.k12.il) decided to go on-line about one year ago. We had a 60-computer Novell network already up and running, and we dedicated one DX4-100 for the job of Internet server. However, in many cases students and teachers wanted (or were required) to write their own WWW pages. At first, that person would write a page, and I would copy it to the Internet server manually. This is a very clumsy solution that worked only at first, and it doesn't allow the user to edit their pages.

At some point I found ncpfs. It was a very experimental project then, but it did most of what I needed it to do. Right now, our Novell server (freud.leyada.jlm.k12.il) is always mounted by our Linux Internet server (www.leyada.jlm.k12.il) as /novell (a cron script checks that this is so, and mounts the server if not). The httpd web server automatically looks for pages in a specific directory inside the Novell hierarchy, which solves the problem. This

directory, say G:\WWW, contains our entire home page. When a student wishes to create a home page, he requests that a directory be opened for him under that directory, say G:\WWW\HOME\JOE. He receives Novell write permission to that directory, and is able to edit HTML files with his favorite web editor. This technique also allows everyone to use DOS and Windows to edit HTML files, which in our case is what the Novell clients run. Therefore, a user edits an HTML file through Windows and checks it with Netscape, while the page is LIVE, since the Linux machine mounts the Novell server.

Shay Rojansky is a 17-year-old high school student and Computer Science student at the Hebrew University of Jerusalem. He works in his high school as a system administrator (mainly Linux) and in the CS institute at the Hebrew University as a lab assistant. You can send him email at roji@cs.huji.ac.il.

Archive Index Issue Table of Contents

Advanced search

The "Virtual File System" in Linux

Alessandro Rubini

Issue #37, May 1997

This article outlines the VFS structure and gives an overview of how the Linux kernel accesses its file hierarchy. The information herein refers to Linux 2.0.x (for any x) and 2.1.y (with y up to at least 18).

The main data item in any Unix-like system is the "file", and a unique path name identifies each file within a running system. Every file appears like any other file in the way it is accessed and modified: the same system calls and the same user commands apply to every file. This applies independently of both the physical medium that holds information and the way information is laid out on the medium. Abstraction from the physical storage of information is accomplished by dispatching data transfer to different device drivers. Abstraction from the information layout is obtained in Linux through the VFS implementation.

The Unix Way

Linux looks at its file system in the same way Unix does—adopting the concepts of super block, inode, directory and file. The tree of files accessible at any time is determined by how the different parts are assembled, each part being a partition of the hard drive or other physical storage device that is "mounted" to the system.

While the reader is assumed to be well acquainted with the concept of mounting a file system, I'll detail the concepts of super block, inode, directory and file.

- The **super block** owes its name to its heritage, from when the first data block of a disk or partition was used to hold meta information about the partition itself. The super block is now detached from the concept of data block, but it still contains information about each mounted file system. The actual data structure in Linux is called **struct super_block** and holds

various housekeeping information, like mount flags, mount time and device block size. The 2.0 kernel keeps a static array of such structures to handle up to 64 mounted file systems.

- An **inode** is associated with each file. Such an “index node” holds all the information about a named file except its name and its actual data. The owner, group, permissions and access times for a file are stored in its inode, as well as the size of the data it holds, the number of links and other information. The idea of detaching file information from file name and data is what allows the implementation of hard-links—and the use of “dot” and “dot-dot” notations for directories without any need to treat them specially. An inode is described in the kernel by a **struct inode**.
- The **directory** is a file that associates inodes to file names. The kernel has no special data structure to represent a directory, which is treated like a normal file in most situations. Functions specific to each file system type are used to read and modify the contents of a directory independently of the actual layout of its data.
- The **file** itself is associated with an inode. Usually files are data areas, but they can also be directories, devices, fifos (first-in-first-out) or sockets. An “open file” is described in the Linux kernel by a **struct file** item; the structure holds a pointer to the inode representing the file. **file** structures are created by system calls like **open, pipe** and **socket**, and are shared by father and child across **fork**.

Object Orientedness

While the previous list describes the theoretical organization of information, an operating system must be able to deal with different ways to layout information on disk. While it is theoretically possible to look for an optimum layout of information on disks and use it for every disk partition, most computer users need to access all of their hard drives without reformatting, to mount NFS volumes across the network, and to sometimes even access those funny CD-ROMs and floppy disks whose file names can't exceed 8+3 characters.

The problem of handling different data formats in a transparent way has been addressed by making super blocks, inodes and files into “objects”; an object declares a set of operations that must be used to deal with it. The kernel won't be stuck into big **switch** statements to be able to access the different physical layouts of data, and new file system types can be added and removed at run time.

The entire VFS idea, therefore, is implemented around sets of operations to act on the objects. Each object includes a structure declaring its own operations, and most operations receive a pointer to the “self” object as the first argument, thus allowing modification of the object itself.

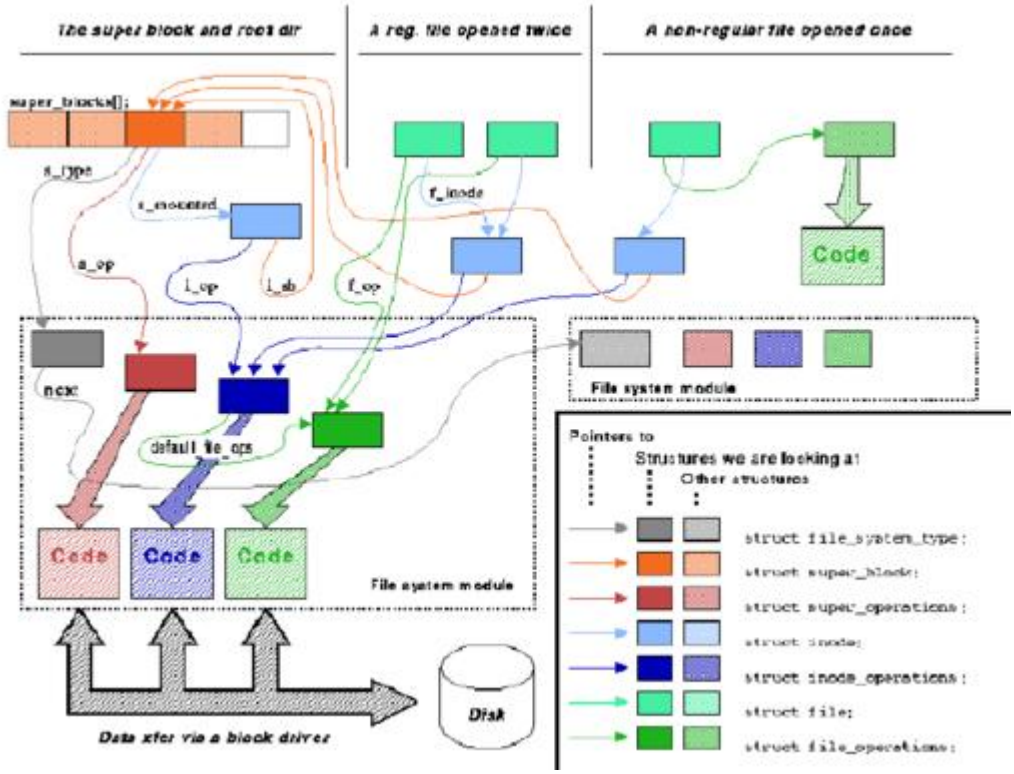
In practice, a super block structure encloses a field **struct super_operations *s_op**, an inode encloses **struct inode_operations *i_op** and a file encloses **struct file_operations *f_op**.

All the data handling and buffering performed by the Linux kernel is independent of the actual format of the stored data. Every communication with the storage medium passes through one of the **operations** structures. The file system type, then, is the software module which is in charge of mapping the operations to the actual storage mechanism—either a block device, a network connection (NFS) or virtually any other means of storing and retrieving data. These modules can either be linked to the kernel being booted or compiled as loadable modules.

The current implementation of Linux allows use of loadable modules for all file system types but root (the root file system must be mounted before loading a module from it). Actually, the **initrd** machinery allows loading of a module before mounting the root file system, but this technique is usually exploited only on installation floppies.

In this article I use the phrase “file system module” to refer either to a loadable module or a file system decoder linked to the kernel.

This is in summary how all file handling happens for any given file system type, and is depicted in Figure 1:



- **struct file_system_type** is a structure that declares only its own name and a **read_super** function. At **mount** time, the function is passed information about the storage medium being mounted and is asked to fill a super block structure, as well as loading the inode of the root directory of the file system as **sb->s_mounted** (where **sb** is the super-block just filled). The additional field **requires_dev** is used by the file system type to state whether it will access a block device: for example, the NFS and **proc** types don't require a device, while **ext2** and **iso9660** do. After the super block is filled, **struct file_system_type** is not used any more; only the super block just filled will hold a pointer to it in order to be able to give back status information to the user (**/proc/mounts** is an example of such information). The structure is shown in Listing 1.

Listing 1

- The **super_operations** structure is used by the kernel to read and write inodes, write super block information back to disk and collect statistics (to deal with the *statfs* and *fstatfs* system calls). When a file system is eventually unmounted, the *put_super* operation is called—in standard kernel wording “get” means “allocate and fill”, “read” means “fill” and “put” means “release”. The **super_operations** declared by each file system type are shown in Listing 2.

Listing 2

- After a memory copy of the inode has been created, the kernel will act on it using its own operations. **struct inode_operations** is the second set of operations declared by file system modules, and is listed below; they deal mainly with the directory tree. Directory-handling operations are part of the inode operations because the implementation of a **dir_operations** structure would bring in extra conditionals in file system access. Instead, inode operations that only make sense for directories will do their own error checking. The first field of the inode operations defines the file operations for regular files. If the inode is a fifo, a socket or a device-specific file operation will be used. Inode operations appear in Listing 3; note the definition of **rename** was changed in release 2.0.1.

Listing 3

- The **file_operations**, finally, specify how data in the actual file is handled: the operations implement the low-level details of **read, write, lseek** and the other data-handling system calls. Since the same **file_operations** structure is used to act on devices, it also includes some fields that only make sense for character or block devices. It's interesting to note that the structure shown here is the structure declared in the 2.0 kernels, while 2.1

changed the prototypes of **read, write** and **lseek** to allow a wider range of file offsets. The file operations (as of 2.0) are shown in Listing 4.

Listing 4

Typical Implementation Problems

The mechanisms to access file system data described above are detached from the physical layout of data and are designed to account for all the Unix semantics as far as file systems are concerned.

Unfortunately, not all file system types support all of the functions just described—in particular, not every type has the concept of “inode”, even though the kernel identifies every file by means of its **unsigned long** inode number. If the physical data accessed by a file system type has no physical inodes, the code implementing **readdir** and **read_inode** must invent an inode number for each file in the storage medium.

A typical technique to choose an inode number is using the offset of the control block for the file within the file system data area, assuming the files are identified by something that can be called a “control block”. The **iso9660** type, for example, uses this technique to create an inode number for each file in the device.

The **/proc** file system, on the other hand, has no physical device from which to extract its data and, therefore, uses hardwired numbers for files that always exist, like **/proc/interrupts**, and dynamically allocated inode numbers for other files. The inode numbers are stored in the data structure associated with each dynamic file.

Another typical problem faced when implementing a file system type is dealing with limitations in the actual storage capabilities. For example, how to react when the user tries to rename a file to a name longer than the maximum allowed length for the particular file system, or when she tries to modify the access time of a file within a file system that doesn't have the concept of access time.

In these cases, the standard is to return **-ENOPERM**, which means “Operation not permitted”. Most VFS functions, like all the system calls and a number of other kernel functions, return zero or a positive number in case of success, and a negative number in the case of errors. Error codes returned by kernel functions are always one of the integer values defined in **<asm/errno.h>**.

Dynamic /proc Files

I'd now like to show a little code to play with VFS, but it's quite hard to conceive of a small enough file system type to fit in the article. Writing a new file system type is surely an interesting task, but a complete implementation includes 39 “operation” functions.

Fortunately enough, the **/proc** file system as defined in the Linux kernel lets modules play with the VFS internals without the need to register a whole new file system type. Each file within **/proc** can define its own inode operations and file operations and is, therefore, able to exploit all the features of the VFS. The method of creating **/proc** files is easy enough to be introduced here, although not in too much detail. “Dynamic **/proc** files” are so named because their inode number is dynamically allocated at file creation (instead of being extracted from an inode table or generated by a block number).

In this section we build a module called **burp**, for “Beautiful and Understandable Resource for Playing”. Not all of the module will be shown because the innards of each dynamic file are not related to VFS.

The main structure used in building up the file tree of **/proc** is **struct proc_dir_entry**. One such structure is associated with each node within **/proc**, and it is used to keep track of the file tree. The default **readdir** and **lookup** inode operations for the file system access a tree of **struct proc_dir_entry** to return information to the user process.

The **burp** module, once equipped with the needed structures, will create three files: **/proc/root** is the block device associated with the current root partition, **/proc/insmod** is an interface to load/unload modules without the need to become root, and **proc/jiffies** reads the current value of the jiffy counter (i.e., the number of clock ticks since system boot). These three files have no real value and are just meant to show how the inode and file operations are used. As you see, **burp** is really a “Boring Utility Relying on Proc”. To avoid making the utility too boring I won't give the details about module loading and unloading, since they have been described in previous *Kernel Korner* articles which are now accessible on the Web. The whole **burp.c** file is available as well from SSC's ftp site.

Creation and destruction of **/proc** files is performed by calling the following functions:

```
proc_register_dynamic(struct proc_dir_entry \
   *where, struct proc_dir_entry *self);
proc_unregister(struct proc_dir_entry *where, \
    int inode);
```

In both functions, **where** is the directory where the new file belongs, and we'll use **&proc_root** to use the root directory of the file system. The **self** structure, on the other hand, is declared inside **burp.c** for each of the three files. The definition of the structure is reported in Listing 5 for your reference; I'll show the three **burp** incarnations of the structure in a while, after discussing their role in the game.

Listing 5

The “synchronous” part of **burp** reduces therefore to three lines within **init_module()** and three within **cleanup_module()**. Everything else is dispatched by the VFS interface and is “event-driven” inasmuch as a process accessing a file can be considered an event (yes, this way to see things *is* unorthodox, and you should never use it with professional people).

The three lines in **ini_module()** look like:

```
    proc_register_dynamic(&proc_root, \
       &burp_proc_root);
```

and the ones in **cleanup_module()** look like:

```
    proc_unregister(&proc_root, \
       burp_proc_root.low_ino);
```

The **low_ino** field is the inode number for the file being unregistered, and has been dynamically assigned at load time.

But how will these three files respond to user access? Let's look at each of them independently.

- **/proc/root** is meant to be a block device. Its “mode” should, therefore, have the **S_IFBLK** bit set, its inode operations should be those of block devices and its device number should be the same as the root device currently mounted. Since the device number associated with the inode is not part of the **proc_dir_entry** structure, the **fill_inode** field must be used. The inode number of the root device will be extracted from the table of mounted file systems.
- **/proc/insmod** is a writable file. It needs its own **file_operations** to declare its “write” method. Therefore, it declares its **inode_operations** that points to its file operations. Whenever its **write()** implementation is called, the file asks *kerneld* to load or unload the module whose name has been written. The file is writable by anybody. This is not a big problem as loading a module doesn't mean accessing its resources and what is loadable is still controlled by root via **/etc/modules.conf**.

- **/proc/jiffies** is much easier; the file is read-only. Kernel versions 2.0 and later offer a simplified interface for read-only files: the **get_info** function pointer, if set, will be asked to fill a page of data each time the file is read. Therefore, **/proc/jiffies** doesn't need its own file operations nor inode operations; it just uses **get_info**. The function uses **sprintf()** to convert the integer **jiffies** value to a string.

Listing 6

The snapshot of a tty session in Listing 6 shows how the files appear and how two of them work. Listing 7, finally, shows the three structures used to declare the file entries in **/proc**. The structures have not been completely defined, because the C compiler fills with zeroes any partially defined structure without issuing any warning (feature, not bug).

Listing 7

The module has been compiled and run on a PC, an Alpha and a Sparc, all of them running Linux version 2.0.x

The **/proc** implementation has other interesting features to offer, the most notable being the *sysctl* interface. This idea is so interesting, and it will need to be covered in a future *Kernel Korner*.

Interesting Examples

My discussion is now finished, but there are many places where interesting source code is available for viewing. Implementations of file system types worth examining:

- Obviously, the "/proc" file system: it is quite easy to look at, because it is neither performance-critical nor particularly fully featured (except the *sysctl* idea). Enough said.
- The "UMSDOS" file system: it is part of the mainstream kernel and runs piggy-back on the "Ms-DOS" file system. It implements only a few of the operations of the VFS to add new capabilities to an old-fashioned file system format.
- The "userfs" module: it is available from both tsx-11 and sunsite under **ALPHA/userfs**; version 0.9.3 will load to Linux 2.0. This module defines a new file system type which uses external programs to retrieve data; interesting applications are the ftp file system and a read-only file system to mount compressed tar files. Even though reverting to user programs to get file system data is dangerous and might lead to unexpected deadlocks, the idea is quite interesting.

- “supermount”: the file system is available on sunsite and mirrors. This file system type is able to mount removable devices like floppies or CD-ROMs and handle device removal without forcing the user to **umount/mount** the device. The module works by controlling another file system type while arranging to keep the device unmounted when it is not used; the operation is transparent to the user.
- “ext2”: the extended-2 file system has been the standard Linux file system for a few years now. It is difficult code, but worth reading for those interested in seeing how a real file system is implemented. It also has hooks for interesting security features like the immutable-flag and the append-only-flag. Files marked as immutable or append-only can only be deleted when the system is in single-user mode, and are therefore secured from network intruders.
- “romfs”: this is the smallest file system I've ever seen. It was introduced in Linux-2.1.21. It's a single source file, and it's quite enjoyable to browse. As its name asserts, it is read-only.

is a wild soul with an attraction for source code. He is a fan of Linus Torvalds and Baden Powell and enjoys the two communities of volunteer workers they have attracted. He can be reached at rubini@linux.it.

Archive Index Issue Table of Contents

Advanced search

Tips from The Answer Guy

James T. Dennis

Issue #37, May 1997

Learn how to block mail your mail and tighten your security from The Answer Guy.

Netscape Mail Block

I need to refuse to accept e-mail from a particular person. How can I configure Netscape and/or CND1.0 to bounce the person's mail back to them? —Mitch, Mobile, Alabama

And the Answer Is…

*I'd use procmail, a little programming language written specifically for processing mail. CND uses procmail as the "local delivery agent" by default. This means that sendmail passes any mail for a local account to procmail, and then lets procmail do the final delivery to your mail box, /var/spool/mail/*your_login_name. At the same time, procmail checks for a .procmailrc file in your home directory, and does some ownership and permissions checks on it for you.

The author of a .procmailrc file can specify a variety of settings and clauses which are called "recipes", and can also modularize the file by using a variety of INCLUDE directives. Here's a simple example that should get you started:

```
:0 hr
* ^From.*spammer.you.despise@spamhaven.com
* !^FROM_MAILER
* !^FROM_DAEMON
* !^X-Loop: ${USERNAME}i@`hostname`"
| (formail -r -A"X-Loop:
${USERNAME}@`hostname`" \
-A"Precedence: junk";\
   echo "Your mail is not welcome here."\
   echo "Please don't mail me again."\
   echo\
   cat ~/your.signature.or.flame
)
```

*The **:0** marks this as a new recipe—each new recipe starts with this line. The **h** and the **r** on that line are flags tell procmail which parts of the message to hand to your action line (i.e., the one that starts with a pipe, |).*

- **h** says: "Give me the header."
- **r** says: "Treat the incoming data as raw."

The **r** flag is given to prevent your response from failing if the sender has failed to put a blank line at the end of his message.

The following four "star" lines in the script are conditions. The first specifies that the header will show that the message is "from" your spammer, that is, your unwanted sender. This address will exactly match any "from" or "From:" line that contains your target e-mail address. The next two lines of the script ensure that you don't respond to daemons and mailers (mailing lists). The last * line, which you should fill in with your user name and host name, ensures that your don't respond to your own response. Those three conditions are included to protect your script from being tricked into undesirable actions. Consider them to be the minimum overhead on any auto-responders that you write.

The next line of the script, which starts with a "|" pipe character, describes the action to take. In procmail there are three types of actions:

1. A file name specifies an **mbox** (elm, pine or mailx compatible) folder in which to store the message.
2. A directory name specifies an **mh** or **mmdf** folder for mail storage. **mh** and **mmdf** use different naming schemes for the messages in their folder directories, but you don't need to worry about this difference unless you use one of their mail user agents.
3. A **!** (bang) line specifies an e-mail address to which the message is to be bounced. A **|** (pipe) line specifies that the message is to be filtered through a local program.

formail is a program that comes with the procmail package. It "formats mail headers". This particular **formail** command formats a "reply" (**-r**) header, and adds two additional header lines—a standard "Precedence: junk" line and a personal "X-" line. The RFC822 spec allows you to use the X- line to embed custom information into a header. It is also in the **formail** command line that you prevent an attack by routing your response back into your own script, i.e., a mail loop.

The **echo** and **cat** statements after the **formail** line provide output that is appended after the mail header and that becomes the body of your response.

You can add additional echo lines or you can create a file and use **cat** to add it here.

If you are new to procmail (which is almost certain given your question—auto-responders are some of the first things that procmail users learn), you may be nervous about breaking something and losing some of your mail. To protect yourself you will want to start your .procmailrc with the following simple recipe:

```
:0 c
fallback
```

*This recipe, if it is the first recipe in the script, appends a copy of every incoming message to a file named **fallback** in your ~/Mail directory by default. You can compare the contents of that folder to your inbox until you are confident that everything is working as you expect.*

Please read the procmail and procmailex (examples) man pages for more details. The author, Stephen van der Berg, has also written an automated mail list management package called SmartList that is highly regarded among people that I know who have used it. I like SmartList much better than Majordomo. —Jim

Dealing with E-mail on a POP3 Server

Is there any way (or any program out there) which will not only get my e-mail from a POP3 server off of one account, but distribute it to multiple users on my system by either the **from:** or **subject:** lines?

Perhaps popclient could get the mail and save to temp. Then a program could go through the saved mail and say, “Hmmm, this mail is from johndoe@linux.org and it goes to root—then the next message is from mike@canoe.net and it goes to Dave.” Is there a program that will do this? —Moe Green, starved@ix.netcom.com

And the Answer Is…

It is possible to write procmail scripts that can take this sort of action for you. Although I don't recommend this approach, I'll tell you how to do it.

The current version of popclient is called fetchmail, because it supports IMAP and some other mail store and forward protocols. The fetchmail default is to fetch the mail from your POP or IMAP server and feed it to the smtpd (sendmail) on your local host. This means that any special processing that would be done by the aliases or .forward files (especially any processing through procmail scripts) will be done automatically.

It is possible to override that feature and feed the messages through a pipe or into a file. It is also possible, using procmail or any scripting language, to parse and dispatch the file. Using anything other than procmail would require that you know a lot about RFC822, the standard for Internet mail headers, and about e-mail in general.

I wrote an article on procmail that appears in February's Linux Gazette, Issue 14. The gist of it is also available on my own mail server, and can be obtained by sending mail to info@starshine.org with a subject of **procmail** or **mailbot**.

The reason I don't recommend using procmail in this way is that it violates the intentions and design of Internet e-mail. A better solution is to find a provider of UUCP (Unix-to-Unix CoPy) services or at least SMTP/MX (Simple Mail Transfer Protocol) services. UUCP is the right way to provide e-mail to disconnected (dial-up) hosts and networks. It was designed and implemented over 25 years ago, and all of the mail systems on the Internet know how to gateway to UUCP sites.

As for SMTP/MX services for disconnected hosts/networks, there are various ways of hacking sendmail and DNS (Domain Name Service) configurations that have been developed in the last few years with a variety of shell scripts and custom programs to support them. All of these methods provide essentially the same services as mail via UUCP over TCP but do not conform to any standard, which means that whatever you learn and configure with one ISP probably won't work with the next one. —Jim

Security Problem

Recently a cracker got into my Linux system on the Internet. He didn't do a lot of damage, but I guess he did turn off system logging, since I couldn't see what he'd done. Now I can't get it working again. Here's what I've done so far:

1. I've made sure that the syslogd program is running using **ps**.
2. I've read the syslogd.conf file to make sure it's logging everything, and where it's going to.
3. I've checked permissions on the log file.
4. I did a **kill -HUP** on the syslogd process, and it writes **restart** to the log.
5. **logger** does nothing when I run it (no log entry, no error).
6. All my C programs that wrote to syslog don't anymore.

Anyone have any good ideas what to do from here? —
Jayjay@shadow.ashpool.com

And the Answer Is…

I do, but they are rather too involved for me to type up tonight. However, I highly recommend that you reinstall the OS and all binaries from scratch whenever you think root has been compromised on your system. I realize this is a time-consuming proposition, but it is the only way to truly be sure.

I also recommend the program tripwire that can be found at ftp.cs.perdue.edu in the COAST archive, and its mirrors.

Please feel free to write me at jimd@starshine.org if you continue to have system security problems.

Sorry to take so long to respond. I've been literally swamped all month. —Jim

More on Security

I found that the cracker had replaced my syslogd with a packet sniffer. I think he had copied the syslogd code and replaced parts of it with his sniffer. It seemed to have some functionality but not a lot...

I also found a SUID version of bash in my /tmp directory. My thought is this is the way he originally got root access. —Jay

Not too surprising. He was probably using a **rootkit**; however, he obviously didn't do a very good job of covering his tracks. You should consider all passwords for all of the systems on the local net to be compromised. Force password changes across the board and consider installing **ssh** or **stelnet**. Both are secure, encrypted replacements to **rlogin/rsh** and **telnet** respectively.

He probably got in through the “Leshka” sendmail bug that allows any shell user to create a root-owned SUID shell in /tmp/ on any system that has an SUID root copy of sendmail (version ~8.6.x to 8.7.x ?). The bug involves sendmail's handling of ARGV[0] and it's subsequent SIGHUP (signal to disconnect) handling. Everyone using earlier versions of sendmail should upgrade to 8.8.3 or later (see http://www.sendmail.org/ for details).

How important are this system and the other systems on the same LAN segment to your business? I'd seriously consider hiring a qualified consultant for a full day risk assessment and audit. Unfortunately, you'll probably pay at least $125/hr for anyone that's worth talking to, and many of the “security consultants” out there are snake oil salesmen, so beware. —Jim

This article was first published in Issue 14 of LinuxGazette.com, an on-line e-zine formerly published by Linux Journal.

Jim Dennis is the proprietor of Starshine Technical Services. His professional experience includes work in technical support, quality assurance and information services (MIS) for software companies like Quarterdeck, Symantec/ Peter Norton Group and McAfee Associates—as well as positions with smaller VARs. He's been using Linux since version 0.99p10 and is an active participant on an ever-changing list of mailing lists and newsgroups. He's just started collaborating on the 2nd Edition for a book on Unix systems administration. Jim is an avid science fiction fan—and recently got married at the World Science Fiction Convention in Anaheim

Archive Index Issue Table of Contents

Advanced search

New Products

M. L. Richardson

Issue #37, May 1997

NetAcquire 3000 Server, Telaxian Shield Firewall System, Willows RT for Tornado and more.

NetAcquire 3000 Server

Real Time Integration, Inc. announced Unix drivers for the NetAcquire 3000, a network data acquisition server, that acquires, processes and updates real-time analog data at over 750,000 samples/second using a standard Ethernet network to communicate. While an off-the-shelf Linux release is not yet available, it is designed to compile under Linux. The NetAcquire 3000 model is priced at $8495.

Contact: Real Time Integration, Inc.,7914 140 Pl. NE, Redmond, WA 98052-4180, Phone: 206-883-7563, Fax: 206-883-0463, E-mail: realtimeint@realtimeint.com, URL: http://www.realtimeint.com/.

Telaxian Shield Firewall System

Network Engineering Technologies, Inc. (N.E.T.) announced Telaxian Shield, a firewall system capable of mirroring the organizational and geographical structure of an entire enterprise. The Telaxian Shield is priced from $7,995 to $11,995, depending on the specific configuration. It is available for Linux.

Contact: Network Engineering Technologies, 1714 Ringwood Ave., San Jose, California 95131, Phone: 408-453-7500, Fax: 408-437-0651, URL: http://www.fireants.com/.

Willows RT for Tornado

Wind River Systems and Willows Software, Inc. introduce Willows RT for Tornado, a new solution for bringing standard Windows software to the real-time embedded market. Products developed using Willows RT are portable

across a wide range of microprocessors including Linux. It is available for $6500 for a single-seat license.

Contact: Wind River Systems, Inc., Alameda, CA 94501, Phone: 800-545-WIND, E-mail: inquiries@wrs.com, URL: http://www.wrs.com/.

Microway Alpha-based Workstations

Microway announced 500 MHz Screamer workstations with 2MB of synchronous SRAM cache. These desktop supercomputers utilize DEC's latest Alpha technology, plus Microway-engineered motherboards and positive pressure SIMM cooling for workstations containing 128MB or more of memory. Also available is Microway's ported and maintained version of Linux for the Alpha. For pricing, contact Microway.

Contact: Microway, P.O. Box 79, Kingston, MA 02364, Phone: 508-746-7341, Fax: 508-746-4678, URL: http://www.microway.com/.

SpellCaster Telecommute/BRI

SpellCaster Telecommunications, Inc. today announced the TeleCommute/BRI, a high-performance, intelligent ISDN Basic Rate (BRI) terminal adapter card for ISA bus personal computers. It is a complete high-speed data and voice communications solution. It is available for $573.

Contact: SpellCaster Telecommunications Inc., Toronto, Canada, Phone: 800-238-0547, E-mail: jdw@spellcast.com, URL: http://www.spellcast.com/.

NovaLink e.prise

NovaLink USA Corp. announced e.prise, an environment for creating and managing web sites for the Internet and Intranet. The basis of the technology is a sophisticated object-oriented content database and user-friendly design. NovaLink's e.prise is available for Linux. Pricing is dependent on number of licenses.

Contact: NovaLink USA Corp., 200 Friberg Parkway, Westborough, MA 01581, Phone: 508-898-2000, Fax: 508-836-4766, E-mail: amazing@novalink.com, URL: http://www.novalink.com/.

PanGlot Multi-lingual E-mail Editor for Linux

PanGlot Software announces the availability of its Linux multi-lingual e-mail editor. With this editor it is possible to use up to seven languages/alphabets simultaneously in a single document. Each language/alphabet has its own individualized keyboard map. Others can be loaded from disk as required. The

FREE e-mail reader can be downloaded from Sunsite or our home page; the $25.00 mailer can be ordered from PanGlot Software .

Contact: PanGlot Software, 6430 North Strahan, El Paso, TX 79932, Phone: 416-297-1927, E-mail: stermole@panglot.com, URL: http://www.panglot.com/.

Reactor 4.1

Critical Mass, Inc. announced Reactor 4.1, the Distributed Application Development Environment. Reactor allows your distributed applications to seamlessly cross Linux/ELF and Win32. It will allow Linux developers to build robust applications targeted for Windows NT and Windows 95, as well as other Unix platforms.

Contact: Critical Mass, Inc., Cambridge, MA, Phone: 617-354-6277, E-mail: farshadi@cmass.com, URL: http://www.cmass.com/reactor/.

Network Technologies Video Switch

Network Technologies Inc. announced a new line of video switches which allow two computers on different platforms to share the same monitor. The SE-SPV-2 allows a PC and a Sun workstation to share a monitor and retails for $280. Each of the available switches comes with a one year warranty, and has an optional remote RMT-2-ST.

Contact: Network Technologies Inc., 1275 Danner Dr., Aurora, OH 44202, Phone: 800-742-8324, E-mail: sales@networktechinc.com, URL: http// www.networktechinc.com/.

FrontPage Server Extensions for Linux

Ready-To-Run Software announced the availability of Microsoft FrontPage Server Extensions for a wide range of Unix web servers including Linux. FrontPage is a web authoring and management tool. For pricing information contact Ready-To-Run Software.

Contact: Ready-To-Run Software, 4 Pleasant St., P.O. Box 2038, Forge Village, MA 01886-5038, Phone: 508-692-9922, Fax: 508-692-9990, E-mail: info@rtr.com, URL: http://222.rtr.com/.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #37, May 1997

Our experts answer your technical questions.

Netscape On MkLinux

I have installed MkLinux on my Mac 6100. I log in and ftp Netscape. When I try to run the executable I get an error message saying I can't run the binary. —Manny Duarte

No MkLinux Binary

As far as I know, Netscape does not support MKLinux. The binary they supply for Linux is compiled for Intel CPUs. It will not work on a Mac. —Bob Hauck, Wasatch Communications Group bobh@wasatch.com

Terminating pppd

I am running kernel 1.2.13 and pppd 2.1.2 on a remote machine and kernel 2.0.18 and pppd 2.2.0 on a local machine to connect to the remote machine by modem. When the modem drops carrier (due to line noise, etc.) the remote pppd process remains active, preventing getty accepting any more connections until pppd is killed.

Is it possible to have the remote pppd terminate automatically when the modem drops the carrier? —Eskinder Mesfin

Running pppd Manually

The **modem** option does exactly that, assuming that your modem is set up to have the DCD signal follow the carrier state and that your cables pass all of the relevant signals through to the serial port. Most modems will operate DCD in the desired mode if you include AT&C1 in the init string.

However, if you are manually running pppd on the remote machine after logging in to a shell, there is a caveat. In that case you need to **exec** pppd rather than simply running it.

If you just run pppd from the command line, the shell, rather than the pppd process, will get the SIGHUP signal when you hang up. The shell will terminate but leave the pppd daemon running. Instead, do **exec pppd**. This will replace the running shell with pppd so that the hang-up signal will work correctly. —Bob Hauck, Wasatch Communications Group bobh@wasatch.com

Generating RARP Requests

I need to set up a large number of Linux boxes as X-terminals. I would like to automate the addressing of these boxes through RARP, preferably. I have had little trouble learning how to make a Linux box answer RARP queries, but I can't seem to make it generate one. Any help you could offer would be appreciated. —George

Enabling RARP

The kernel can do it. You can boot a kernel, have it generate a RARP request, and then use the machine that answers as an NFS server. It can do the same using the BOOTP protocol as well. These options must be enabled during the **make config** of the kernel.

I suspect what you want is something like what a Sun workstation does, which is to get a kernel from the network starting with a RARP request. That kind of thing can be done only with special ROMs available only for certain Ethernet cards. You can likely find information at http://sunsite.unc.edu/linux. I recommend using small hard disks or booting a kernel from floppy to an NFS server. It is much easier to work with. —Donnie Barnes, Red Hat Software redhat@redhat.com

Plug and Play Modems

I used Windows 95 before Linux; that's why I got a plug and play modem. It works with Windows 95 but it does not work in Linux. Why? —Stou Sandalski

Two Types of Plug and Play

There are two different types of internal modems that fall into the plug and play category. The first is a standard modem that is simply configured at boot time to determine what COM port it will provide. The second is called a WinModem.

A WinModem does not have a UART, which is what makes a normal serial port tick. Instead, they knock down the price of the modem by $10-$20 and eliminate this normally necessary component of any serial port or modem. It is replaced with a software driver that emulates the UART's functionality.

If you have, or think you have, such a modem, your only hope would be to ask the manufacturer whether they support Linux for that modem. If not, the modem cannot be used under Linux. If you do not have a WinModem, you should be fine, provided you set the modem up correctly.

No two modems are identical. Your best bet is to consult the manufacturer. In most cases, if you do not have a product that will work under Linux, they will provide an upgrade at a very low cost. —Chad Robinson, BRT Technical Services Corporationredhat@redhat.com

Users Cannot Change Password

I have installed shadow-ina-box-1.2 and all the accounts that I created after the installation get the following error when they try to change their password:

```
homepage:~$passwd
Changing password for user_name
The password for user_name cannot be changed
```

Is there a solution to this or will I have to revert to an open password file? — Mike Pelley

Changing Permissions

I'd guess that one of two things is happening here. Either your passwd binary doesn't have the right permissions, or you are still using your old non-shadow passwd binary.

In order for passwd to make changes to the passwd file, it must be suid root. To check this, try doing **ls -l `which passwd`**. It should print something like this:

```
-r-s--x--x   1 root     bin          3152 May  4  1994 /usr/bin/passwd
```

*The important things are the **s** in the first column and the **root** in the third column. If you don't see the **s**, do a **chmod u+s `which passwd`** as root. If the file isn't owned by root (the **root** in the third column), do **chown root `which passwd`**.*

Before you do all of that though, double check that what you are running really is the binary that shadow-ina-box installed. Do **which passwd** and make sure

that's the right passwd binary. —Steven Pritchard, Southern Illinois Linux Users Group stever@silug.org

Converting Text To PostScript

I print over a network setup. My problem is that when I print text files I cannot control the font size, and lines are cut off at the end. Is there any utility that will help me convert a text file to PostScript in any font size, because I have no problem printing PostScript. —Eskinder Mesfin

Multiple Solutions

GNU enscript is a drop-in replacement for the enscript program. Enscript converts ASCII files to PostScript and writes the generated output to a file or sends it directly to the printer.

It is available from: prep.ai.mit.edu:/pub/gnu/enscript-1.4.0.tar.gz. —Rory Toma, WebTV Networks rory@corp.webtv.net

The nenscript program does what you want. It has numerous options to control the font, paper size, lines per page, number of copies, and so forth.

You might want to look into the magicfilter utility. This is a nifty little program that allows you to transparently print almost any kind of file to any reasonable printer. It installs as a print filter and uses some heuristics to determine the file type and work accordingly. I got my copy from Sunsite. —Bob Hauck, Wasatch Communications Group bobh@wasatch.com

Try the apsfilter (aps 4.9.1) available on every linux-mirror. —Klaus Franken, S.u.S.E. GmbHkfr@suse.de

Archive Index Issue Table of Contents

Advanced search