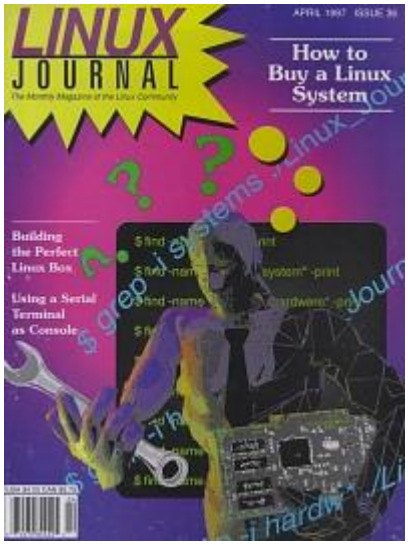
Linux Journal Issue #36/April 1997



Features

Serial Terminal as Console *by Francesco Conti*
Sans video card, sans keyboard, sans monitor: the amazing headless Linux box.

Building the Perfect Box: How to Design Linux Workstation *by Eric S Raymond*
These days, it's possible to put together a decent personal Unix Platform for less than $2,000 US.

Thread-Specific Data and Signal Handling in Multi-Threaded Applications *by Martin McCarthy*
This second part of a series on Multi-threading deals with how to use C programs with one of the POSIX packages available for Linux to handle signals and concurrent threads

News & Articles

Creating Animations with POV-Ray *by Andy Vaught*
The /proc File System and ProcMeter *by Andrew M. Bishop*
Somebody Still Uses Assembly Language? *by Richard A. Sevenich*

Reviews

Product Review Applixware 4.2 for Linux *by Gary Moore*
Book Review Active Java and Exploring Java *by Danny Yee*

Serial Terminal as Console

Francesco Conti

Issue #36, April 1997

Complete instructions for setting up a cheap headless Linux box.

A frequently asked, but never completely answered, question in the comp.os.linux.* newsgroups and other Linux mailing lists, is the one about cheap, headless Linux boxes. It seems that many people need to install Linux boxes without a video card, a monitor or a keyboard.

A cheap response to this problem is to use a serial terminal (Wyse or Ampex, for example) as the main Linux console. This cuts the cost of a keyboard, a video card and a monitor. I've done this very thing on my second computer, an old 486 VLB, by using a Wyse 60 terminal.

Linking a terminal to your computer's serial port is not at all difficult. You can easily follow the instructions in the Serial-HOWTO and in the inittab(5) and agetty(8) man pages. Here's a short summary.

First, you must use a null modem cable.

Second, insert the following line in your /etc/inittab file, if you're using agetty. Other getty programs, like getty_ps, use a different syntax.

```
ID:RUNLEVELS:respawn:/sbin/agetty -L SPEED TTY TERM
```

where:

- ID = a two character identifier, e.g., s1 or s2
- RUNLEVELS = Runlevels in which the terminal must be active
- SPEED = serial port speed
- TTY = tty port name relative to the /dev directory
- TERM = value to be used for the TERM environment variable

My machine's /etc/inittab has the following line:

```
s2:12345:respawn:/sbin/agetty -L 9600 ttyS1 vt100
```

for a serial terminal on the /dev/ttyS1 port (COM2 for DOS users), with a port speed of 9600 BPS and vt100 terminal emulation (which seems to run better than native Wyse 60 mode).

Finally, restart init with the command **init q**.

If you correctly followed these three simple steps, you should see the login prompt on your terminal screen. You can log in and work on your machine in the same way you can when you're actually on the console or telneting from a remote host.

Kernel Messages

The messages the kernel shows at boot time are always directed to your main console (tty1). If you turn on your headless box, you can only wait for the login on the terminal, which means losing those precious messages. You can see them by using the dmesg command, but usually you need them before the login shell comes up.

There are other messages on your console: those generated by the scripts in the /etc/rc.d directory, and from scripts run at boot and shutdown time. How can you really know that “the system is halted” if you can't read it on a monitor?

You must patch the /usr/src/linux/drivers/char/console.c program in your kernel source tree. It's not a complex kernel hack. You can follow these three simple steps.

First, define the CONFIG_SERIAL_ECHO symbol at program start:

```
#define CONFIG_SERIAL_ECHO
```

Second, modify the address of the terminal serial port (only if you're using a port different from that defined by default) looking for the following line:

```
#define SERIAL_ECHO_PORT        0x3f8   /* COM1 */
```

In my machine I've changed that line to:

```
#define SERIAL_ECHO_PORT        0x2f8   /* COM2 */
```

Third, rebuild your kernel and reboot: you should see on your terminal screen the kernel messages during your system's hardware devices probe.

Please note that these steps work for a 2.0.0 kernel, not on 1.2.13. I haven't yet had time to try other kernels. The console.c patch is necessary for all Linux ports except the one for Alpha, which contains it in the **make config** with the following option:

```
Echo console messages on /dev/ttyS1
```

Messages from /etc/rc.d/rc.*

To show these messages on your terminal, you can append **> TTY** to every line of these files that contains the command **echo**. TTY is the terminal serial port (the same one used in /etc/inittab serial terminal line).

LILO Configuration

If you want to choose among more than two kernel images, you have to modif the LILO configuration file, /etc/lilo.conf.

Complete instructions for seeing the LILO prompt on serial terminals can be found in the /usr/doc/lilo/README file (look for the SERIAL option). Here are two steps to do that correctly.

First, edit the /etc/lilo.conf file and insert a SERIAL option line after the BOOT option line:

```
serial=SERIAL_LINE,SPEED PARITY BITS
```

where:

```
SERIAL_LINE = 0 (com1)
              1 (com2)
              2 (com3)
              3 (com4)
SPEED = serial port speed
PARITY = n (= none)
         o (= odd)
         e (= even)
BITS = bits in a character (8 or 7)
```

Please note that there are no spaces between the **SPEED**, **PARITY** and **BITS** parameters. These must be equal to the ones defined in your terminal setup. Here's the line used by my machine's LILO:

```
serial=1,9600n8
```

This line means COM2 at 9600 BPS, no parity, 8 bits per character.

Second, execute the **lilo** command to update your system's configuration.

Using the SERIAL option, LILO sets a two second delay (the same as when you put a **delay = 20** line in the lilo.conf file) before booting the default kernel

image. During this pause, you can interrupt the boot process and get the LILO prompt by sending a break to the terminal as you press the SHIFT key on your main console.

Conclusion

At last, your serial terminal can be used as a real system console. I think that the only thing you can't do is the CTRL-ALT-DEL reset—except, perhaps, with certain terminal emulations. If you're lucky, you can find one of these terminals cheaply, maybe even free, from a company upgrading its hardware.

Franceso Conti can be reached by e-mail at fconti@iper.net.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Building the Perfect Box: How To Design Your Linux Workstation

Eric S. Raymond

Issue #36, April 1997

This article is a guide to building capable Linux workstations from cheap generic PC hardware.

Most of the good things about Linux flow from the fact that it makes a full-featured Unix accessible on inexpensive hardware. Accordingly, there's a huge amount of documentation and folk knowledge in the Linux community about how to get people who already have cheap hardware to use Linux on it. Up to now there hasn't been much advice available on how to acquire cheap hardware that is well-matched to Linux, for someone who already knows Linux.

At today's prices, it's possible to put together a terrific personal Unix platform for less than $2,000 US. If you're prepared to go mail-order, shop carefully and make a few minor tradeoffs, you can do it for $1,500 or even less. But beware. If you buy as though for a DOS/Windows box, you won't get the best value or performance. Linux works its hardware harder than Unix does, and configurations that are marginal under DOS/Windows can cause problems under Linux.

In this article, we'll develop a recipe for a cheap but capable Linux workstation. While developing it, we'll discuss the recipe choices in some detail, and see how to avoid common pitfalls that can cause you grief.

We are going to stick to Intel hardware in this article. Alphas are fast and have that wonderful 64-bit architecture, and SPARCs too have earned their fans. However, I think PC hardware is still overall the most cost-effective—cheapest to buy, easiest to get serviced and best-tested with Linux. And, given the relative sizes of the respective markets, PC hardware seems likely to hold its lead for years yet.

For more detail on this subject, organized in a reference rather than narrative format, surf to my PC-Clone UNIX Hardware Buyer's Guide at http://www.ccil.org/~esr/clone-hw-guide/contents.html. I've been maintaining this document and its FAQ ancestor for longer than Linux has existed, and have been running Unix on PC hardware since shortly after it first became possible in the late 1980s.

What To Optimize

Most people think of the processor as the most important choice in specifying any kind of personal-computer system. Our first lesson in building Linux boxes is this: for Linux, the processor type is nearly a red herring. It's far more important to specify a capable system bus and disk I/O subsystem.

One important reason for this is precisely because PC systems are marketed in a way that presents processor speed as a primary figure of merit. The result is that the development of processor technology has naturally gotten pushed harder than anything else, and off-the-shelf PCs have processors that are quite overpowered relative to the speed of everything else in the system. Your typical PC these days has spare CPU-seconds it will never use, because the screen, disk, modem and other peripherals can't be driven fast enough to tax it.

If you're already running Linux, you may find it enlightening to keep top(1) running for a while as you use your machine. Notice how seldom the CPU idle percentage drops below 90%.

It's true that after people upgrade their motherboards, they often report a throughput increase. But this is often due more to other changes that go with the processor upgrade, such as improved cache memory or an increase in the system bus's clocking speed, i.e., enabling data to get in and out of the processor faster.

The unbalanced, processor-heavy architecture of PCs is hard to notice under DOS and Windows 3.1, because neither OS hits the disk very much. But any OS that uses virtual memory and keeps lots of on-disk logs and other transaction states is a different matter—it will load the disk more heavily and will suffer more from the imbalance.

Linux is in this category, and I'd guess Windows NT and OS/2 are too. Assuming you're buying for Linux on a fixed budget, it makes sense to trade away some excess processor clocks to get a faster bus and disk subsystem.

The truth is that **any** true 32-bit processor now on the market is more than fast enough for your disks under a typical Linux-like load, even if it's a lowly 386/25. Your screen, if you're running X, can be a bit more demanding—but even a

486/50 will let you drag Xterm windows around like paper. And that's a lot slower than the cheapest new desktop machine you'll be able to find by the time this article hits paper.

So buy a fast bus. And especially, buy fast disks. How does this translate into a recipe? Like this:

- **Don't** bother with the latest Pentium Pro whizbang 300mHz super-scorcher with a cooling fan bigger than it is.
- **Do** get a PCI-bus machine.
- **Do** get a SCSI controller.
- **Do** get the fastest SCSI disks you can afford.

Buying PCI will get you maximum bus throughput, and makes sense from several other angles as well. The doggy old ISA bus is clearly headed for extinction at this point, and you don't hear much about its other competitors (EISA, VESA local-bus video or MCA) anymore. With PCI now being used in Macintoshes and Alphas as well as all high-end Intel boxes, it's clearly here to stay, and a good way to protect your investment in I/O cards from rapid obsolescence.

The case for SCSI is a little less obvious, but is still compelling. For starters, SCSI is still at least 10-15% faster than EIDE running flat out. Furthermore, EIDE is still something of a “jerry-rig”. Like Windows, it's layered over an ancestral design (ST-512) that's antiquated and prone to failure under stress. SCSI, on the other hand, was designed from the beginning to scale up well to high-speed, high-throughput systems. Because it's perceived as a “professional” choice, SCSI peripherals are generally better engineered than EIDE equivalents. You'll pay a few dollars more, but for Linux the cost is well repaid in increased throughput and reliability.

For the fastest disks you can find, pay close attention to seek and latency times. The former is an upper bound on the time required to seek to any track; the latter is the maximum time required for any sector on a track to come under the heads, and is a function of the disk's rotation speed.

Of these, seek time is more important and is the one manufacturers usually quote. When you're running Linux, a one millisecond faster seek time can make a substantial difference in system throughput. Back when PC processors were slow enough for the comparison to be possible (and I was running System V Unix), it was easily worth as much as a 30mHz increment in processor speed. Today the corresponding figure would be higher.

What Processor Should I Buy?

We just got through with a lengthy explanation of why processor speed isn't that important. But nobody is going to buy a 386/25 (or even 386/50) at this point; if nothing else, you want to get a newer motherboard so you can put this week's flavor of RAM module on it. And who knows? Maybe you **will** end up doing real-time 3D graphics or nuclear-explosion modeling or one of the handful of applications that can really strain your processor.

So for all you processor-speed junkies out there who want to be able to wave around megahertz figures like gearheads bragging abut the compression ratios in their hot-rods, here's a simple rule:

- **Do** buy one or two levels lower than commercial state of the art.

As of December 1996, if you look at a typical clone-maker's advertisement, you'll see that the top three systems are a Pentium Pro, a Pentium 166 and a Pentium 133. The rule of thumb tells us to skip the Pentium Pro, consider the Pentium 166, and look seriously at the 133.

Why? Because of the way manufacturers' price-performance curves are shaped. The top-of-line system is generally boob bait for corporate executives and other people with more money than sense. Chances are the system design is new and untried—if you're at the wrong point in the technology cycle, the chip may even be a pre-production sample, or an early production step with undiscovered bugs, like the infamous Pentium FDIV problem. You don't need such troubles. Better to go with a chip/motherboard combination that's been out for a while and is well known. It's not like you need the extra speed, after all.

Besides, if you buy one of these gold-plated systems, you're only going to kick yourself three months later when the price plunges by 30%. Further down the product line there's been more real competition, and the manufacturer's margins are already squeezed. There's less room for prices to fall, so you won't watch your new toy lose street value so fast. Its price will still drop, but it won't plummet sickeningly.

Again, bear in mind that the cheapest processor you can buy new today is plenty fast enough for Linux. So if dropping back to a Pentium 90 or 75 will bring you in under budget, you can do it with no regrets.

One Disk or Two?

At December 1996 prices, there's really no reason to consider buying less than a 1-gigabyte disk. This is a convenient size, because “install everything” on most Linux distributions will lay out more than 540MB, but less than 1GB of stuff.

If you can afford 2GB, the natural thing to think about is buying a 2GB disk instead. But personally, I like a configuration with two 1GB disks better—one “system” disk and one “home” disk. There are several good reasons for this kind of setup. Most of them come down to the fact that you are quite a bit less likely to trash two disks at once than you are to trash a single one.

A lot of us do Linux upgrades every three months or so. Wouldn't it give you a warm, comforting feeling during your next one to be able to dismount your “home” disk in advance and **know** that there's no way the upgrade can possibly step on your personal files?

Or let's suppose you have a fatal disk crash. If you have only one disk, goodbye Charlie. If you have two, maybe the crashed one was your system disk, in which case you can buy another and mess around with a new Linux installation knowing your personal files are safe (see above). Or maybe it was your home disk; in that case, you can still run and do recovery stuff and basic Net communications until you can buy another home disk and restore it from backups (you **did** keep backups, right?).

You can even tune your disk configuration for performance this way. SCSI controllers can interleave requests to different disks, so your swapper and other system daemons will be able to use scratch files on the system disk at the same time your applications are using files on the home disk. Thus, you may find you actually get faster throughput with two smaller disks than one big one.

To get the most leverage from this effect, choose your system disk for access speed and your home disk for capacity. In December 96 I would ideally choose a 1GB fast system disk and a 2GB home disk.

Monitor And Video

First, buy your monitor. We won't go into detail about this here because the issues aren't at all specific to Linux—you can find good guidance in any DOS-related buyer's guide. There's not a whole lot of price variance among functionally equivalent monitors, since it's a mature commodity technology, so the basic question is, “how many square inches of screen can you afford?”

This is one of the areas where pinching pennies is not a good idea. You're going to be looking at your monitor for hours on end, and using the screen real estate

constantly. Buy the best quality, largest screen you possibly can—it will be worth it. I personally shelled out $2,000 for a 21-inch monitor in January 1996. Though I have no regular income and this represented a significant portion of my bank account, I have never regretted it.

The reasons not to pinch pennies are also reasons why you should actually see the monitor you're contemplating before you buy it. A factory flaw like serious edge mis-convergence or a tilted yoke is not a happy thing to discover just after you've cut a check.

You may want to consider looking for a repaired or reconditioned monitor with a warranty. These are often as good as new and much cheaper.

Next, buy your card. The major issue here is matching the card to the capacity of your monitor—you don't want to pay for more card than your screen can use, and you don't want to buy too cheap a card and find it can't drive your monitor at its maximum capability.

So once you've specified your monitor, find a video card with a maximum video bandwidth equal to or just slightly higher than the monitor's. That way you know your video system is properly balanced with a minimum of wasted capacity.

There is a fair amount of price variance among equivalent video cards, so shop aggressively here. If you're on a budget, one easy thing to trade away is bit depth. Manufacturers like to include 16- and 24-bit “photographic” color as sizzle in their advertisements, but unless you're doing something like specialty photocomposition work or medical graphics, you'll never use more than 256. So you can settle for 8-bit color.

The days when XFree86 seriously constrained your choice of video card are long past. Just about anything you can buy in a clone system should work fine these days. If you're in doubt about whether the card is supported, surf over to http://www.xfree86.org/ and check out their compatibility list.

Easier Choices

Now we've got a good handle on the most important choices, disks, bus, processor and video. The rest is easier, and less dependent on the peculiarities of Linux.

Next in importance is your CD-ROM drive, since you'll almost certainly be installing Linux from it. You have a SCSI system, so get a SCSI CD-ROM. That's pretty much the end of spec, as SCSI CD-ROMs are a very generic item. The only

significant difference among drivers is their speed—6x, 8x, 10x or up. It's hard to find 2x or 4x speeds anymore.

Again, bear in mind that you probably don't need the latest and greatest. High-speed CD-ROMS are really designed for people playing CD-ROM games or other applications involving image and sound archives. If you're doing the Linux thing, chances are you'll primarily use CD-ROMs that are code archives. Your average transfer size will be small and an apparent speed of 6x or even 4x is quite satisfactory. So here's a place to cut costs by buying well behind the leading edge.

Next, consider a backup device. This is another place where spending extra money pays. Cheap tape drives are unreliable, noisy and have agonizingly slow transfer speeds. It's no fun to listen to what sounds like a blender dicing celery for twenty minutes while your disk is backed up, so with the cheap drives you'll quickly find you're backing up less often than you really should.

The worst are the QIC mini-cartridge drives, including the new Travan technology. They're also the cheapest, and thus, exactly what clone makers tend to bundle and salespeople tend to push. Avoid them. Quarter-inch QIC drives are less nasty, mainly because they have higher transfer speed and get done quicker; also they're usually engineered better for reliability. But you really want to pay for a DAT or DLT drive.

Of course, buy 16MB of memory, unless you really like a text-only console—X is not comfortable when memory is tight. Having lots of free memory will also improve your virtual-memory performance. Fortunately, with RAM as cheap as it is now this is unlikely to bust your budget.

You'll also want a three-button serial mouse. I happen to like the three-button Logitech MouseMan and its kin—just the thing for a hacker's chronically cluttered desktop. Your mileage may vary.

You'll want a modem, of course. 28.8 is recommended for speediest possible net surfing. “How to Buy a Modem” could be an article in itself; we won't try to cover it here.

The rest is basically frills and freebies. You can get a sound card if you like, though under Linux you're not likely to use it for anything but playing Doom. Every system sold today has the requisite two floppy drives and two serial ports and one parallel port.

That about does it for basic hardware. Later on we'll look at some actual system configurations.

Some Pitfalls To Avoid.

Don't buy jumperless peripheral cards, whether of the newer variety called "Plug'n'Play" or the older kind that requires a DOS utility to poke registers in the card at bootup time. These will cause you no end of grief. Plug'n'Play isn't yet supported under Linux as I write, and it would be totally nasty to have to boot DOS first every time you want to run Linux. A lot of these cards don't even hold their settings across a warm boot.

You need to be extra-vigilant about this when buying. The tiny, reptilian brains of most computer salespeople cannot seem to encompass the existence of clone-box OSs other than DOS or Windows. They think boot-time setup utilities are just fine and jumperless cards are the best things since sliced bread, because they are easier to set up and a whole fifty cents cheaper. So they'll push jumperless cards at you with glee and abandon. Foil them, or you'll suffer for it later.

Don't buy so-called "WinModems" or anything that advertises "RPI" or "Rockwell Peripheral Interface" on the box. These are ways for manufacturers to save a few bucks on firmware at your expense; they won't work without driver software that runs only under Windows.

Don't get stuck with a 2-button mouse. Specifying a three-button model is an easy detail to overlook when filling out your order.

How To Buy

When I originally launched the Buyer's Guide years ago, the major distribution channels for PCs were business-oriented storefront dealerships and mail order. The dealerships had, and still have, high overheads and higher prices. Accordingly, I recommended mail order.

I still like mail order, especially for techies on a tight budget. Publications like *Computer Shopper* and their web site at http://www.netbuyer.com/, are a great way to get a feel for prices, and these days most mail-order outfits with enough cash to advertise on glossy paper are good risks. The online version of my Buyer's Guide (see the URL at the top of this article) has details on how to protect yourself when buying through the mail.

These days, though, I'm also a fan of computer superstores—outfits like CompUSA and Circuit City that sell hundreds of machines a day out of warehouse-sized premises packed to the ceiling with discounted hardware. These obviously have more overhead than mail-order outfits, but their price premium over mail-order is small. They make back a lot of their margin on

computer games and small accessories like mouse pads, cables and floppy disks.

So, if you shop carefully and don't fall for one of their name-brand “prestige” systems, you can get prices comparable to mail order with the comfort of knowing there's a trouble desk you can drive back to in a pinch. Also, you **can** see your monitor before you buy.

When To Buy

In a market where prices are dropping fast, it's always tempting to wait just another month or two to buy your hardware, because you know it will then be less expensive.

A good way to cope with this problem is to configure your system on paper, get a couple of initial estimates, then set a target price, below the lowest one, at what you're willing to pay. Then watch and wait. When the configuration cost hits your target price, place your order.

The advantage of this method is that it requires you to settle in your mind, well in advance, what you're willing to pay for what you're getting. That way, you'll buy at the earliest time you should, and won't stress out too much afterwards as it depreciates.

The Recipe File

Let's look at some sample components, keeping in mind the guidelines we've developed. All prices are from the December 1996 *Computer Shopper* or their web site at http://www.netbuyer.com/. Check this site out—its search facilities are pretty good. Prices and vendors have been selected to represent what's generally available out there.

- A: P55TV PCI with Pentium 166 and AHA 2940 SCSI on board: $740 from Treasure Chest Peripherals.
- B: Same P55TV with Pentium 133: $520
- C: Same P55TV with Pentium 100: $420
- D: Same P55TV with Pentium 75: $380
- E: 2 16MB 4x32 SIMMs: $208 from Memory Etc.
- F: Seagate 1050MB Hawk 2XL: $350 from Insight Direct.
- G: Seagate 2149MB Hawk 2XL: $480 from Insight Direct.
- H: “Jumbo” Mid-Tower case + power supply: $40 from Sam's Computers.
- I: 6x SCSI CD-ROM from MicroXperts: $160

- J: Hitachi CM1587 15" 1024x768 color monitor: $375 from Automated Tech Tools, Inc.
- K: ASTVision 7L 17" 1280x1024 color monitor: $400 from Tredex
- L: Trident Microsystems 9440 1MB PCI SVGA (1024x768): $31 from Hi-Tech USA
- M: Cirrus Logic CL-GD54M30 1MB PCI 1280x1024: $39 from MicroXperts
- N: Dalco 1.44 floppy: $43.50 from Dalco Electronics
- O: Dalco 1.2 floppy: $58.50 from Dalco Electronics
- P: Seagate MS4000R-SB (4GB SCSI internal DAT): $300 from Global Computer Supplies
- Q: Americomp 3-button mouse: $5 from Americomp

Now let's put these together into sample system configurations.

First, the deluxe system (A+E+F+G+H+I+K+M+N+O+P+Q): $2,824. That's pretty far off our $2,000 target, but now let's strip away as much as we can.

If we drop the 1.2MB floppy (you'll never use it), the secondary disk, and then downgrade to a 15" monitor with 1024x768 resolution, and drop back to a Pentium 75 (D+E+F+H+I+J+L+N+O+P+Q), suddenly the price is just $1,892.50.

Now, when you reflect that vendors of desktop systems buy in volume and get the parts up to 40% cheaper than you can even by mail-order, you can see that coming in under $1500 with a Pentium-75 system shouldn't really be difficult at all.

Even without a system vendor's volume discount, moving back up to a Pentium 100 (C+E+F+H+I+J+L+N+O+P+Q) makes our price $1932.50, still below our $2000 target. The parts list above is just intended as an example—it wouldn't actually be a good idea to build your box by separately assembling pieces like that. It's better and usually cheaper to go through a system vendor.

There are several advantages to using a system vendor. One is the vendor's ability to buy in volume and carve its margins mainly out of the volume discount it gets from parts suppliers. This is especially important for big-ticket items like the motherboard and disks. Another is expert assembly. A third is the pre-shipment burn-in. And then, of course, there's the warranty—a very reassuring thing to have in case your brand-new machine succumbs in spite of that burn-in.

Once you've designed your configuration, you should get quotes from two or three different system vendors. Any vendor who can't generate quotes for a

custom configuration, or resists giving a quote without a buy commitment, is not worth your time—find another.

Questions To Ask Your Vendor

The weakest guarantee you should settle for should include:

- 72-hour burn-in to avoid sudden death Also, try to find out if they do a power-cycling test and how many repeats they do; this stresses the hardware much more than steady burn-in.
- 30-day money-back guarantee. Watch out for fine print that weakens this with a re-stocking fee or limits it with exclusions.
- 1 year parts and labor guarantee (some vendors give 2 years).
- 1 year of 800 number tech support (many vendors give lifetime support).

Additionally, many vendors offer a year of on-site service free. You should find out who they contract the service to. Also, be sure the free service coverage area includes your site; some unscrupulous vendors weasel their way out with “some locations pay extra,” which translates roughly as “through the nose if you're further away than our parking lot.”

If you're buying from a dealership or superstore, find out what they'll guarantee beyond the above. If the answer is “nothing”, go somewhere else.

Ask your potential suppliers what kind and volume of documentation they supply with your hardware. You should get, at minimum, operations manuals for the motherboard and each card or peripheral. Skimpiness in this area is a valuable clue that they may be using no-name parts from Upper Baluchistan, which is not necessarily a red flag in itself, but should prompt you to ask more questions.

There are various cost-cutting tactics a vendor can use which bring down the system's overall quality. Here are some good questions to ask:

- Is the memory zero-wait-state? One or more wait states allows the vendor to use slower and cheaper memory, but will slow down your actual memory subsystem throughput. This is a particularly important question for the cache memory.
- If you're buying a factory-configured system, does it have FCC certification? While it's not necessarily the case that a non-certified system is going to spew a lot of radio-frequency interference, certification is legally required—and becoming more important as clock frequencies climb. Lack of that FCC sticker may indicate a fly-by-night vendor, or at least one in danger of being raided and shut down.

- Are the internal cable connectors keyed, so they can't be put in upside down? This doesn't matter if you'll never, ever **ever** need to upgrade or service your system. Otherwise, it's pretty important. Vendors who fluff this detail may be quietly cutting other corners.

After You Take Delivery

Your configuration is custom and involves slightly unusual hardware. Therefore, keep a copy of the configuration you wrote down, and check it against both the invoice and the actual delivered hardware. If there is a problem, calling your vendor back right away will maximize your chances of getting the matter settled quickly.

Then install your Linux and have fun.

Eric S. Raymond has been interested in Unix since 1973, on the Internet since 1976, among the leading Emacs hackers since 1983, and maintainer of the Jargon File since 1990. He is listed as a Linux core developer. When not hacking, he is likely to be found reading science fiction, playing jazz flute, or practicing telepathy with his girlfriend's cat. His web page is at http://www.ccil.org/~esr.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Thread-Specific Data and Signal Handling in Multi-Threaded Applications

Martin McCarthy

Issue #36, April 1997

Here are the answers to questions about signal handling and taking care of global data when writing multi-threaded programs.

Perhaps the two most common questions I'm asked about multi-threaded programming (after “what is multi-threaded programming?” and “why would you want to do it?”) concern how to handle signals, and how to handle cases where two concurrent threads use a common function that makes use of global data, and yet the two threads need thread-specific data from that function. By definition, global data includes static local variables which are in truth a kind of global variable. In this article I'll explain how these questions can be dealt with in C programs using one of the POSIX (or almost POSIX) multi-threading packages available for Linux. I live in hope of the day when the most common question I'm asked about multi-threaded programming is, “Can we give you lots of money to write this simple multi-threaded application, please?” Hey—I can dream, can't I?

All the examples in this article make use of POSIX compliant functionality. To the best of my knowledge at the time I write this, there are no fully POSIX-compliant multi-threading libraries available for Linux. Which of the available libraries is best is something of a subjective issue. I use Xavier Leroy's LinuxThreads package, and the code fragments and examples were tested using version 0.5 of this library. This package can be obtained from http://pauillac.inria.fr/~xleroy/linuxthreads. Christopher Provenzano has a good user-level library, although the signal handling doesn't yet match the spec, and there were still a number of serious bugs the last time I used it. (These bugs, I believe, are being worked on.) Other library implementations are also available.

Information on these and other packages can be found in the comp.programming.threads newsgroup and (to give a less than exhaustive list):

- http://www.mit.edu:8001/people/proven/pthreads.html
- http://www.aa.net/~mtp/PCthreads.html
- ftp://ftp.cs.fsu.edu/pub/PART/PTHREADS

Thread-specific data

As I implied above, I use the term “global data” for any data which persists beyond normal scoping rules, such as static local variables. Given a piece of code like:

```
void foo(void)
{
        static int i = 1;
        printf( "%d\n", i );
        i = 2;
}
```

the first call to this function will print the value 1, and all subsequent calls will print the value 2, because the variable **i** and its value persist from one invocation of the function to the next, rather than disappearing in a puff of smoke as a “normal” local variable would. This, at least as far as POSIX threads are concerned, is global data.

It is commonly said (I've said it myself) that using global data is a bad practice. Whether or not this is true, it is only a rule of thumb. Certainly there are situations where using global data can avoid creating artificial circumstances. The previous article (*Linux Journal* Issue 34) explained how threads can share global data with careful use of mutual exclusion (**mutex**) functions to prevent one thread from accessing an item of global data while another thread is changing its value. In this article I will look at a different type of problem, using a real example from a recent project of mine.

Consider the case of a virtual reality system where a client makes several network socket connections to a server. Different types and priorities of data go down different sockets. High priority data, such as information about objects immediately in the field of view of the client, is sent down one socket. Lower priority data such as texture information, background sounds, or information about objects which are out of the current field of view, is sent down another socket to be processed whenever the client has available time. The server could create a collection of new threads every time a new client connects to the server, designating one thread for each of the sockets to be used to talk to each of the clients. Every one of these threads could use the same function to send a lump of data (not a technical term) to the client. The data to be sent details of the client it is to be sent to, the priority and type of data to be sent could all be

held in global variables, and yet each thread will make use of different values. So how do we do it?

As a trivial example, suppose the only global data which our lump-sending function needs to use is an integer that indicates the priority of the data. In a non-threaded version, we might have a global integer called **priority** used as in Listing 1.

In the multi-threaded version we don't have a global integer, instead we have a global key to the integer. It is through the key that the data can be accessed by means of a number of functions:

1. **pthread_key_create()** to prepare the key for use
2. **pthread_setspecific()** to set a value to thread-specific data
3. **pthread_getspecific()** to retrieve the current value

pthread_key_create() is called once, generally before any of the threads which are going to use the key have been created. **pthread_getspecific()** and **pthread_setspecific()** never return an error if the key that is used as an argument has not been created. The result of using them on a key which has not been created is undefined. Something will happen, but it could vary from system to system, and it can't be caught simply by using good error handling. This is an excellent source of bugs for the unwary. So our multi-threaded version might look like Listing 2.

There are a few things to note here:

1. The implementation of POSIX threads can limit the number of keys a process may use. The standard states that this number must be at least 128. The number available in any implementation can be found by looking at the macro **PTHREAD_KEYS_MAX**. According to this macro, LinuxThreads currently allows 128 keys.
2. The function **pthread_key_delete()** can be used to dispose of keys that are no longer needed. Keys, like all “normal” data items, vanish when the process exits, so why bother deleting them? Think of key handling as being similar to file handling. An unsophisticated program need not close any files that it has opened, as they will be automatically closed when the program exits. But since there is a limit to the number of files a program can have open at one time, the best policy is to close files not currently being used so that the limit is not exceeded. This policy also works well for key handling, as you may be limited in the number of thread-specific data keys a process may have.
3. **pthread_getspecific()** and **pthread_setspecific()** access thread-specific data as **void*** pointers. This ability can be used directly (as in Listing 2), if the

data item to be accessed can be cast as type **void***, e.g., an **int** in most, but not necessarily all, implementations. However, if you want your code to be portable or if you need to access larger data objects, then each thread must allocate sufficient memory for the data object, and store the pointer to the object in the thread-specific data rather than storing the data itself.

4. If you allocate some memory (using the standard function **malloc()**, for instance) for your thread-specific data, and the thread exits at some point, what happens to the allocated memory? Nothing happens, so it leaks, and this is bad. This is the situation where the extra parameter in the **pthread_key_create()** function comes into use. This parameter allows you to specify a function to call when a thread exits, and you use that function to free up any memory that has been allocated. To prevent a waste of CPU time, this destructor function is called only in the case where a thread has made use of that particular key. There's little point in tidying up for a thread that has nothing to be tidied. When a thread exits because it called one of the functions **exit()**, **_exit()** or **abort()**, the destructor function is not called. Also, note that **pthread_key_delete()** does not cause any destructors to be called, that using a key that has been deleted doesn't have a defined behavior, and that **pthread_getspecific()** and **pthread_setspecific()** don't return any error indications. Tidy up your keys carefully. One day you'll be glad you did. So a better version of our code is Listing 3.

Some of this code might look a little strange at first sight. Using **pthread_getspecific()** to store a thread specific value? The idea is to get the memory location this thread is to use, and then the thread specific value is stored there.

Even if global data is anathema to you, you might still have good use for thread-specific data. In particular, you might need to write a multi-threaded version of some existing library code that is also going to be used in a non-threaded program. A good simple example is making a version of the standard C libraries fit for use by multi-threaded programs. That friend of all C programmers, **errno**, is a global variable that is commonly set by library functions to indicate what went wrong during a function call. If two threads call functions which both set errno to different values, at least one of the threads is going to get the wrong information. This is solved by having thread-specific data areas for **errno**, rather than one global variable used by all threads.

Signal Handling

Many people find signal handling in C to be a bit tricky at the best of times. Multi-threaded applications need a little extra care when it comes to signal handling, but once you've written two programs, you'll wonder what all the fuss was about—trust me. And if you start to panic, remember—deep, slow breaths.

A quick re-cap of what signals are. Signals are the system's way of informing a process about various events. There are two types of signals, synchronous and asynchronous.

Synchronous signals are a result of a program action. Two examples are:

1. **SIGFPE**, floating-point exception, is returned when the program tries to do some illegal mathematical operation such as dividing by zero.
2. **SIGSEGV**, segmentation violation, is returned when the program tries to access an area of memory outside the area it can legally access.

Asynchronous signals are independent of the program. For example, the signal sent when the user gives the **kill** command.

In non-threaded applications there are three usual ways of handling signals:

1. Pretend they don't exist, perhaps the most common policy, and quite adequate for lots of simple programs—at least until you want your program to be reliable and useful.
2. Use **signal()** to set up a signal handler—nice and simple, but not very robust.
3. Use the POSIX signal handling functions such as **sigaction()** and **sigprocmask()** to set up a signal handler or to ignore certain signals—the “proper” method.

If you choose the first option, then signals will have some default behavior. Typically, this default behavior will cause the program to exit or cause the program to ignore the signal, depending on what the signal is. The latter two options allow you to change the default behavior for each signal type—ignore the signal, cause the program to exit or invoke a signal-handling function to allow your program to perform some special processing. Avoid the use of the old-style **signal()** function. Whether you're writing threaded or non-threaded applications, the extra complications of the POSIX-style functions are worth the effort. Note that the behavior of **sigprocmask()**, which sets a signal mask for a process, is undefined in a multi-threaded program. There is a new function, **pthread_sigmask()**, that is used in much the same way as **sigprocmask()**, but it sets the signal mask only for the current thread. Also, a new thread inherits the signal mask of the thread that created it; so a signal mask can effectively be set for an entire process by calling **pthread_sigmask()** before any threads are created.

In a multi-threaded application, there is always the question of which thread the signal will actually be delivered to. Or does it get delivered to all the threads?

To answer the last question first, no. If one signal is generated, one signal is delivered, so any single signal will only be delivered to a single thread.

So which thread will get the signal? If it is a synchronous signal, the signal is delivered to the thread that generated it. Synchronous signals are commonly managed by having an appropriate signal handler set up in each thread to handle any that aren't masked. If it is an asynchronous signal, it could go to any of the threads that haven't masked out that signal using **sigprocmask()**. This makes life even more complicated. For instance, suppose your signal handler must access a global variable. This is normally handled quite happily by using **mutex**, as follows:

```
void signal_handler( int sig )
{
        ...
        pthread_mutex_lock( &mutex1 );
        ...
        pthread_mutex_unlock( &mutex1 );
        ...
}
```

Looks fine at first sight. However, what if the thread that was interrupted by the signal had just itself locked **mutex1**? The **signal_handler()** function will block, and will wait for the mutex to be unlocked. And the thread that is currently holding the mutex will not restart, and so will not be able to release the mutex until the signal handler exits. A nice deadly embrace.

So a common way of handling asynchronous signals in a multi-threaded program is to mask signals in all the threads, and then create a separate thread (or threads) whose sole purpose is to catch signals and handle them. The signal-handler thread catches signals by calling the function **sigwait()** with details of the signals it wishes to wait for. To give a simple example of how this might be done, take a look at Listing 4.

As mentioned earlier, a thread inherits its signal mask from the thread which creates it. The **main()** function sets the signal mask to block all signals, so all threads created after this point will have all signals blocked, including the signal-handling thread. Strange as it may seem at first sight, this is exactly what we want. The signal-handling thread expects signal information to be provided by the **sigwait()** function, not directly by the operating system. **sigwait()** will unmask the set of signals that are given to it, and then will block until one of those signals occurs.

Also, you might think that this program will deadlock, if a signal is raised while the main thread holds the mutex **sig_mutex**. After all, the signal handler tries to grab that same mutex, and it will block until that mutex comes free. However, the main thread is ignoring signals, so there is nothing to prevent another thread from gaining control while the signal handling thread is blocked. In this

case, **sig_handler()** hasn't caught a signal in the usual, non-threaded sense. Instead it has asked the operating system to tell it when a signal has been raised. The operating system has performed this function, and so the signal handling thread becomes just another running thread.

Differences in Signal Handling between POSIX Threads and LinuxThreads

Listing 4 shows how to deal with signals in a multi-threading environment that handles threads in a POSIX compliant way.

Personally, I like the kernel-level package “LinuxThreads” that makes use of Linux 2.0's **clone()** system call to create new threads. At some point in the future, the **clone()** call may implement the **CLONE_PID** flag which would allow all the threads to share a process ID. Until then each thread created using “LinuxThreads” (or any other packages which chooses to use **clone()** to create threads) will have its own unique process ID. As such, there is no concept of sending a signal to “the process.” If one thread calls **sigwait()** and all other threads block signals, only those signals which are specifically sent to the **sigwait()**-ing thread will be processed. Depending on your application, this could mean that you have no choice other than to include an asynchronous signal handler in each of the threads.

Summary

Thread specific data is easy to use—far easier than many people's first experiences may suggest. In a way, this ease of use is a disadvantage, since very often there are more elegant solutions to a problem. But in times of need, thread specific data is your friend.

On the other hand, signal handling in anger can be a little hairy. Anyone who thinks otherwise has overlooked something—either that or they're far too clever for their own good. Make life easier for yourself by consigning all the handling of asynchronous signals to one thread that sits on **sigwait()**.

Martin McCarthy discovered multi-threaded programming while writing the server for a high-speed, multi-user, distributed, virtual-reality system. Of course, he only took that job so that he could squeeze as many buzzwords into his job description as possible. He can be reached at marty@ehabitat.demon.co.uk.

Archive Index Issue Table of Contents

Advanced search

Creating Animations with POV-Ray

Andy Vaught

Issue #36, April 1997

This article is an introduction to animation using Persistence of Vision ray tracing to create a mailbox that doesn't just sit there.

Silicon Graphics workstations come with a mail notification program called “mailbox”, that informs the window system user if any mail awaits. Instead of displaying a simple bitmap like the xbiff program, mailbox draws a real-looking mailbox with a red flag which smoothly rises when it detects new mail. When the user clicks the mouse in the window, the door of the mailbox opens to reveal either empty space or waiting letters. In the latter case, the user's favorite mail program is run.

In this article, I will trace part of the development of a free version of mailbox—building and animating the mailbox. Since SGI workstations contain special graphics-processing chips, the images generated by mailbox are calculated on the fly. On lesser endowed machines, the solution is to generate and store the frames beforehand and simply write them sequentially to the screen. Other examples of similar animations can be found in most of the more popular browsers as well as many web pages.

The natural tool to use is a ray tracing program. While using a paint program will certainly do the job, it would take quite a bit of work to produce smooth animation as well as consistent sizes of the various objects in what is really a very simple scene. The Persistence Of Vision (POV) ray tracing program is one of the oldest free ray tracers around. Its development parallels that of Linux in that it has also been written by a large, unpaid and widely scattered development team.

Ray tracing gets its name from the method it uses to construct an image. For each pixel in the image, a line is calculated in space back along the path of light to see where the light composing that pixel came from, and from there what color it must be. If the ray being traced backwards intersects, say, a mirror

surface, then another ray is traced to see where the reflected ray came from. If a ray comes from a solid object, then another calculation is done to see how that spot is illuminated by the surrounding light sources.

From this simple description, one can see that ray tracing involves many mathematical operations for each pixel being rendered. Math coprocessors found on 486DX systems and beyond will increase performance by a factor of about thirty. As distributed, POV comes with X and SVGALIB versions. One of the great advantages of running POV under Linux as opposed to one of the other operating systems (DOS, Windows or Macintosh) is the ability to do something else while waiting for the finished image. When running POV under XFree86, start your server in a 16-bit color mode using the command:

```
startx -- -bpp 16
```

You won't regret it. If you are running another program that uses lots of colors, 8-bit mode will end up looking really bad.

POV produces an image by interpreting a simple programming language. Since its only purpose is to define various properties of objects and lights, there are no subroutines, loops or if-tests. While the sparseness of the POV language may seem restrictive, it has the effect of keeping the design of the program focused on ray tracing rather than its extension language. A wide variety of third-party programs exist whose ultimate output is a POV-readable text file.

This article describes POV version 3.0, which although still in beta test as of this writing, has nifty features that make simple things much easier to do as well as providing more support for animation. POV 2 came with a good tutorial that has gotten better in 3.0, which now even comes with an HTML version. Rather than try to top the POV tutorial, I will simply present the source code used to construct the mailbox animations.

Running POV

Running POV without arguments produces on-line help on standard error—130 lines of it, so a pipe to **less** is needed. All POV's arguments can be specified in a file called povray.ini. Command-line arguments override this initialization file. A "+" before an argument turns that option on, "-" turns it off. The most frequently used arguments are **D**, which displays the image as it is being rendered; **P**, which pauses when the image is done; and **+I*filename***, which causes *filename* to be interpreted by POV. POV can output files in PPM, PNG and its own Targa format. These formats are common enough to allow conversion to other formats.

Making an Animation

POV works by interpreting a text file supplied by the user. The file describes the positions and sizes of objects, what their surfaces look like and where the lights and the camera are positioned. The structure of the language is a keyword followed by modifiers enclosed in curly braces. Some keywords require specific modifiers, while others don't require any. Sometimes a modifier will have modifiers of its own, also enclosed in curly braces ({ }).

The amount of planning needed to build an image is directly related to its complexity. Complex images should be built in the same manner as a complex program—by first getting a broad outline to work, and then filling in the details. In our case, the mailbox is fairly simple. We define the z direction to be height above the ground, which is at z=0. The post the mailbox rests on will be located at x=0 and y=0 to make things easy. See Figure 1 for a simple plan of the mailbox.

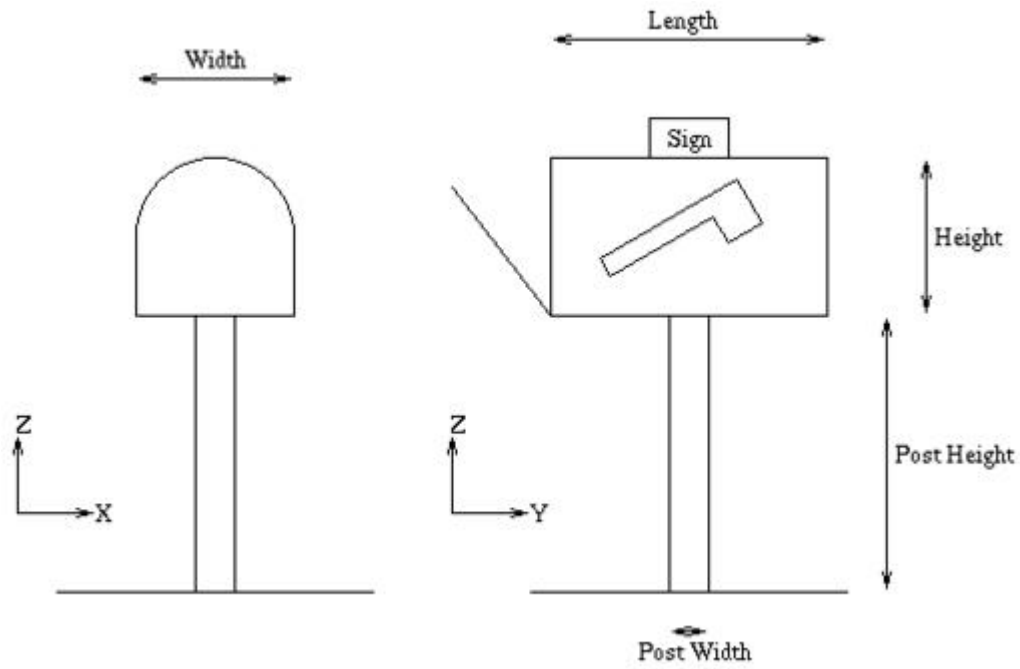


Figure 1. Mailbox Plan

The first part of the mailbox program has the following lines:

```
#include "colors.inc"
#include "shapes.inc"
#include "textures.inc"
declare POST_WIDTH = 1.5
declare POST_HEIGHT = 10.0
declare MB_WIDTH = 3.0
declare MB_LENGTH = 6.0
declare MB_HEIGHT = 3.0
```

If you program in C, you know that the #include followed by a filename causes that file to be inserted in lieu of the #include. POV comes with many include files that provide predefined things. In this case, we're loading color definitions

with symbolic names that are easy to use. We include the shapes file, because it contains the definition for the cylinder we need for the top of the mailbox, and the texture file, which allows us to specify the surface appearance.

The include files are mainly composed of comments and declare directives, which attach a symbolic name to a POV construct. The only thing we're using them for at the moment is to define a couple of constants that give the dimensions of our mailbox in POV's three-dimensional space. This makes changing the dimensions of the mailbox very easy to do. Later, we'll declare the sides and ends of the mailbox, so that we can reuse them easily.

We define the lights and camera next:

```
light_source { <10.0, 0.0, 25.0> color White }
light_source { <0.0, 0.0, POST_HEIGHT+0.95*MB_HEIGHT> Gray50 }
camera {
 location <7.0, 7.0, 13.0>
 sky z
 look_at <0.0, 0.0, POST_HEIGHT+MB_HEIGHT/2>
}
```

In our image, we have two lights. The first one is well above the mailbox and off to one side, the same side as the camera is positioned. The **light_source** keyword is followed by the position and color of the light. The second light is actually inside the mailbox, near the top. Since the only other light doesn't shine into the interior of the box, the interior ends up being a black cave unless we put a light there. Since we want a very small light, we choose the color Gray50 which is halfway (50%) between black and white.

The two main parameters that specify the camera are its location and its point of view. In our case, the camera is at about the same height as the box and off at an angle, so that we can see an end and a side at the same time. The **sky** keyword specifies which direction is up, i.e., at the top of the image. By default, POV assumes the y-direction is up. The z here is actually a shorthand way of specifying the vector <0,0,1>—you can define any direction as up.

Now that lights and camera have been defined, we can start putting things into our little world:

```
background { color SkyBlue }
plane {
 z,0
 color Green
}
```

The **background** keyword is used to specify what color is generated if a light ray does not end up intersecting any object at all. We define this to be a color named SkyBlue. POV is capable of generating much more complicated backgrounds with varying shades of blue, clouds and fog effects.

The **plane** keyword specifies an infinite plane—in our case, the ground. The orientation of a plane can be specified by what is called its normal vector. A normal is the direction sticking straight out of the plane, which for this picture is the z direction. The second parameter is a single number that specifies how far from the origin (the point <0,0,0>) the plane is. Since the ground is at z=0, this parameter is zero.

Building from the ground up, the first object is a wooden post:

```
box {
 < -POST_WIDTH/2.0, -POST_WIDTH/2.0, 0.0 >,
 < POST_WIDTH/2.0, POST_WIDTH/2.0, POST_HEIGHT >
 pigment { DMFWood4 }
}
```

A box is specified by two positions of two opposite corners. The box is always lined up with the x, y and z axes—to get other orientations, the box must be rotated. To get the wood surface, we specify a predefined pigment found in the textures.inc file. The color of a surface is technically part of a pigment which includes more information about the surface, but POV lets us omit the **pigment** keyword when just specifying a color.

Now that we've reached the mailbox itself, we shift the plane a bit to make things easier. Instead of building the mailbox on top of the post (z=POST_HEIGHT), we'll build the mailbox on the ground (z=0), and move the completed box on top of the post later.

```
declare mb_side = polygon { 5,
 < 0.0, -MB_LENGTH/2.0, 0.0 >,
 < 0.0, MB_LENGTH/2.0, 0.0 >,
 < 0.0, MB_LENGTH/2.0, MB_HEIGHT/2.0 >,
 < 0.0, -MB_LENGTH/2.0, MB_HEIGHT/2.0 >,
 < 0.0, -MB_LENGTH/2.0, 0.0 >
}
declare mb_bottom = polygon { 5,
 < -MB_WIDTH/2.0, -MB_LENGTH/2.0, 0.0 >,
 < MB_WIDTH/2.0, -MB_LENGTH/2.0, 0.0 >,
 < MB_WIDTH/2.0, MB_LENGTH/2.0, 0.0 >,
 < -MB_WIDTH/2.0, MB_LENGTH/2.0, 0.0 >,
 < -MB_WIDTH/2.0, -MB_LENGTH/2.0, 0.0 >
}
```

As we mentioned earlier, we're declaring the pieces with the intention of putting them together later. Polygon is a Greek word meaning “many sides”. The first number tells POV how many points in space specify the polygon. For some reason, POV requires that the first point equal the last point. If it doesn't, POV will close the polygon automatically and issue a warning.

The discerning reader will notice we've left out a pigment/color specification. It will be added at the end, so that all of the sides get the same pigment and we don't have to retype the same specification.

Now that we've done some simple shapes, we can do some more complicated things, like the ends of the mailbox:

```
declare mb_end = union {
polygon { 5,
 < -MB_WIDTH/2.0, 0.0, 0.0 >,
 < MB_WIDTH/2.0, 0.0, 0.0 >,
 < MB_WIDTH/2.0, 0.0, MB_HEIGHT/2.0 >,
 < -MB_WIDTH/2.0, 0.0, MB_HEIGHT/2.0 >,
 < -MB_WIDTH/2.0, 0.0, 0.0 >
}
disc {
 < 0.0, 0.0, MB_HEIGHT/2.0 >, y, MB_HEIGHT/2.0
}
}
```

A POV **union** is a collection of one or more objects bound together into a single object. In the case of the mailbox ends, we have a rectangle and a disc that overlap each other. The position of the disc is specified by the location of its center. The orientation is determined by its normal vector (think of it as the direction of an axis). The last parameter is the radius of the disc. Since they overlap perfectly, the disc and rectangle come together seamlessly.

The most complicated object to build is the half-cylinder that forms the top of the mailbox:

```
declare mb_top = intersection {
 box {
  < -2,-1, 0 >
  < 2, 1, 2 >
  pigment { color red 1.0 green 1.0 blue 1.0
        filter 1.0 }
 }
 Cylinder_Y
 scale <MB_WIDTH/2.0, MB_LENGTH/2.0, MB_WIDTH/2.0>
}
```

An **intersection** works similarly to a union, except the final result consists of a surface that is common to all elements of the section. If we'd said **intersection** instead of **union** for the mailbox ends, we would have ended up with a half-disc. In the current problem, we want half of a cylinder.

Cylinder_Y is actually defined in the shapes.inc in terms of the POV **quadric** primitive. The result is an infinite cylinder of radius 1.0 along the y-axis. We chop this off by intersecting the cylinder with a box. This works except for one final detail—the result is a solid half-cylinder as opposed to the outer surface. The opaque flat surfaces come from our bounding box.

The solution to this problem is possible only on a computer—we make the box invisible by giving it a special color. Instead of a predefined color name following the **color** keyword, we specify the amount of red, blue and green to use. The last keyword, **filter**, reverses the whole interpretation of the color so that instead of reflecting light, the surface passes light. If we were to change the command to **color red 1.0 filter 1.0**, we would end up with a surface that

passed only red light. Since our color has the maximum values of red, green and blue, all light is passed.

When the final scene is rendered with the door open, a careful inspection reveals that the result is slightly less than perfect. If you're looking for it, the end of the cylinder is darker than the lower part. The reason for this is a tradeoff POV makes between speed and accuracy.

Recall that a pixel's color is calculated by bouncing a ray around from the viewer until it hits a light source. Inside the box, lots of bouncing is going on and at some point POV has to give up on hitting a source, since each bounce reduces the amount of light being transmitted. The problem is that our invisible surface is still counted as a surface and POV gives up too soon. This is remedied by setting the **max_trace_level** global variable higher. The optimum value can be determined by setting a very high value, then looking at the statistics printed by POV at the end of the run. In the current scene, a maximum of 16 bounces were needed, so we set it to 20 with the following line:

```
global_settings { max_trace_level 20 }
```

After the complicated half-cylinder, the flag of the mailbox seems trivial by comparison. The dimensions of the various parts of the flag are also in Figure 2.

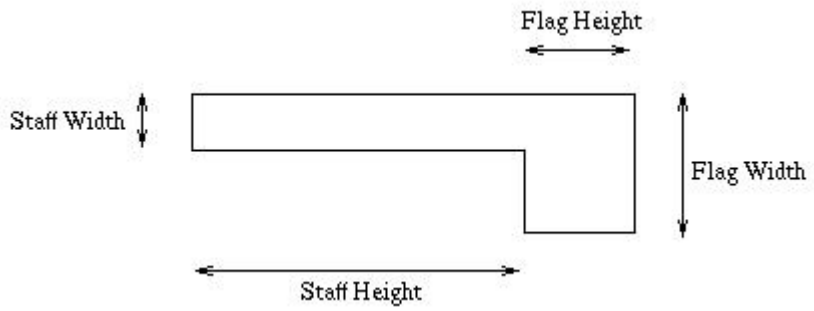


Figure 2. Mailbox Flag

```
declare STAFF_HEIGHT = 3.0
declare STAFF_WIDTH = 0.30
declare FLAG_HEIGHT = 1.0
declare FLAG_WIDTH = 1.5
declare mb_flag = polygon { 7,
 < 0.0, 0.0, 0.0 >,
 < 0.0, -STAFF_HEIGHT, 0.0 >,
 < 0.0, -STAFF_HEIGHT, -FLAG_WIDTH+STAFF_WIDTH >,
 < 0.0, -STAFF_HEIGHT+FLAG_HEIGHT,
        -FLAG_WIDTH+STAFF_WIDTH >,
 < 0.0, -STAFF_HEIGHT+FLAG_HEIGHT, -STAFF_WIDTH >,
 < 0.0, 0.0, -STAFF_WIDTH >,
 < 0.0, 0.0, 0.0 >
 pigment { color Red }
 finish { ambient 0.85 }
}
```

The mailbox flag has a different **finish** than the rest of the mailbox, so we specify it here. The finish determines how light bounces off of the surface in

question. In particular, the **ambient** parameter specifies how much of the ambient light to reflect to the camera. Ambient light is a global parameter that defaults to white light. The 0.85 specifies “lots”, so we end up with a fluorescent red flag that glows even when all other lights are turned off.

The last little piece of the mailbox is a sign on top giving the address.

```
declare sign = union {
text { ttf "timrom.ttf" "Andy" 0.05, 0.0
 pigment { Red }
 finish {
  ambient 0.5
  diffuse 0.5
 }
 translate <0.32, 0.175, 0.0>
}
polygon { 5,
 < 0.0, 0.0, 0.0 >,
 < 3.0, 0.0, 0.0 >,
 < 3.0, 1.0, 0.0 >,
 < 0.0, 1.0, 0.0 >,
 < 0.0, 0.0, 0.0 >
 pigment { White }
}
}
```

Again, we build the sign on the ground and move it into position later. The main reason for this is that text objects are rendered at a fixed place, which is flat on our ground (z=0). Before POV 3.0, text had to be laboriously constructed from a union of polygons. POV 3.0 is capable of reading and displaying Adobe TrueType fonts. Most letters are about one unit high and about half a unit wide. Since POV's world is a three-dimensional world, text objects have a thickness, which we've set to 0.05. This is big enough that the letters stick out from the background rectangle, yet small enough that the thickness is not obvious. The last parameter we've set to zero is a number that specifies an offset between each character beyond the usual spacing.

The downside of text objects is that centering and figuring out how large the text should be is a matter of trial and error. The **translate** keyword modifies an object by moving it in a particular direction, in this case to the center of the following rectangle.

Finally, we put the whole mailbox together. We place everything in a union so that we can translate the whole structure to the top of the post:

```
union {
 object { mb_bottom }
 object {
  mb_side translate < -MB_WIDTH/2.0, 0.0, 0.0>
 }
 object {
  mb_side translate < MB_WIDTH/2.0, 0.0, 0.0>
 }
 object {
  mb_end
  translate < 0.0, -0.5*MB_LENGTH, 0.0 >
 }
```

```
  object {
   mb_end
   rotate < -90.0*clock, 0.0, 0.0 >
   translate < 0.0, 0.475*MB_LENGTH, 0.0 >
  }
```

The first part of the union puts the bottom, sides and ends together. The only new thing in this section is the **rotate** keyword which rotates the front end of the mailbox. The vector after the **rotate** keyword gives the rotation angle in degrees about the x, y and z axes. The **clock** variable is a number between zero and one that is used when a multiple-frame animation is being rendered. Instead of rendering a single frame, the number of frames to be drawn is given on the command line, and the variable **clock** is zero for the first frame and one for the last. The result is a series of images with the door smoothly rotating open. For a single frame, **clock** is set to zero.

```
 object {
   mb_top
   translate < 0.0, 0.0, MB_HEIGHT/2.0 >
  }
  object {
   sign
   scale 0.9
   rotate <90, 0, -90>
   translate <0, 1.5, MB_HEIGHT>
  }
  object {
   mb_flag
   rotate <-90*clock, 0.0, 0.0>
   translate < 0.01+MB_WIDTH/2.0, MB_LENGTH/2.95,
         MB_HEIGHT/2.25 >
  }
  texture {
   pigment { color Silver }
   normal { bumps 0.1 scale 0.01 }
   finish {
    ambient 0.2
    brilliance 6.0
    reflection 0.5
   }
  }
  translate < 0.0, 0.0, POST_HEIGHT>
 }
```

The last section places the mailbox's top, sign and flag into place. The final **texture** keyword is used to give a texture to all the surfaces that don't have one yet—surfaces that already have textures are unaffected. The brilliance parameter controls how light reflects from the surface as a function of the angle at which it strikes the surface. Higher values make the surface appear more metallic, which is what we want. The **reflection** keyword causes reflection to occur, with 1.0 being a mirror surface and 0.0 being a black non-reflective surface.

Of course, the last thing to do is actually run POV to render the images. Running POV in animation mode involves adding a command-line switch **+KFF*n*** where *n* is the number of frames to produce. The output filenames are appended with 01, 02, etc., corresponding to the frame number. For long animations, subsets of the entire sequence can easily be done.

Although the real project isn't done yet, it helps to view the output as it really might be seen. Taking the easy way out, a short Tcl/Tk script is provided (Listing 1) that will display a simple animation forward and backward at a mouse click. Being interpreted, it isn't fast, but it was quick to write and it works. An alternative to this script is to use the GIFMerge program to merge separate GIF images into a single GIF that can be displayed by one of the major web browsers or by XAmin. POV does not write GIF files, but many converters are available.

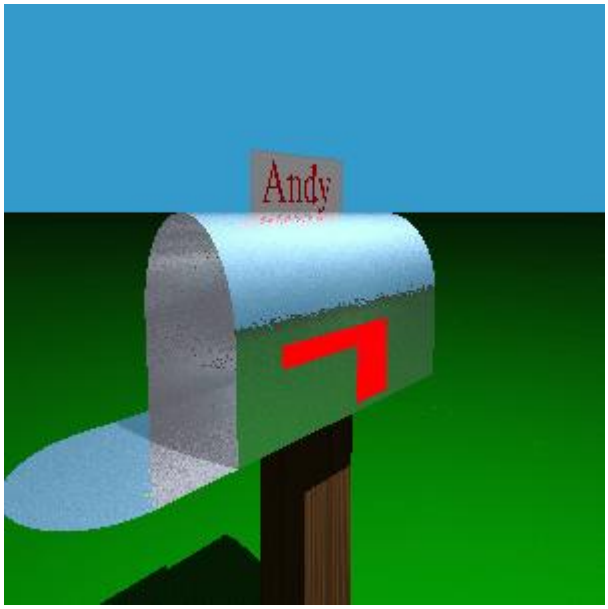


Figure 3: Finished Mailbox

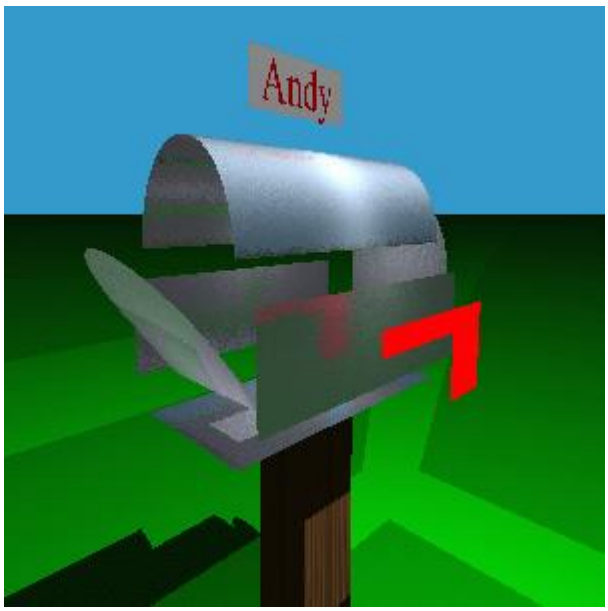


Figure 4: Exploded View of Mailbox

More Information

A wealth of information exists on POV-Ray. The best place to start is the POV web site, http://www.povray.org/. Although the site has recently been

downsized, follow the link to the back issues of POVZINE, a webzine devoted to POV. A POV CD-ROM is available, as well as several books on ray tracing, some specific to POV. GIFMerge has a really neat home page (containing compiled binaries) at http://www.iis.ee.ethz.ch/~kiwi/GIFMerge/.

Andy Vaught is currently a PhD candidate in computational physics at Arizona State University, and has been running Linux since 1.1. He enjoys flying with the Civil Air Patrol as well as skiing. He can be reached at andy@maxwell.la.asu.edu.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

The /proc File System And ProcMeter

Andrew M. Bishop

Issue #36, April 1997

You may rely on your electronic Rolodex to organize your life, but Linux uses the /proc file system.

The /proc file system is a part of Linux that most people have not investigated deeply—perhaps may have never heard of. Like the kernel itself, it is a vital part of a Linux system. Yet its contents and its function are a mystery to most users. If the kernel is the brain of the operating system then the /proc file system is its personal organizer.

In this article I will describe the /proc file system—what it is, and how it can be used. There is also a description of the program ProcMeter that uses the /proc file system to display useful information.

What Is the /proc File System?

First of all, the /proc file system is not a real file system; it is a virtual file system without the physical presence that a disk or a tape has. The most common file system you use is the collection of files on the disk. The disk stores the data without regard to meaning, and the file system (e.g., the Linux ext2fs system) makes sense of the data. The file system organizes the data as directories and files for the user. Another common file system is the Network File System (NFS), which makes files on remote computers accessible.

All file systems are managed by the Linux kernel, which maps the data on the device into a usable form. The user-level programs that access the file system do not need to know how or where the data is actually stored. When a program reads from a file, the kernel manipulates the appropriate device to obtain the data. When a program accesses one of the /proc files there is no device; instead, the kernel supplies the information from its internal state. The files exist only while there is a program actually looking at them.

The /proc file system is a feature which Linux inherited from one of its Unix ancestors. There are two main dialects of Unix in popular usage: System V and BSD. The history of these two are not important here, except that System V contains a /proc and BSD does not.

What Is in This File System?

Everything that is happening in Linux. Every single program that is running, the entire contents of memory, the internal workings of the kernel—all the processes currently running on the system are contained in the /proc file system. proc is an abbreviation for process.

The most interesting files in /proc are listed below. *This list was compiled from kernel version 1.2.13; other versions will be different.* This is not a complete list, but contains only those files whose contents are obvious to casual browsers. A full description of the files can be read in the Linux kernel source code—a task not for the faint-hearted.

The contents of /proc are completely dependent on the processor architecture. (For example, the file /proc/cpuinfo is available only for ix86 processors.) The different types of hardware with which the kernel must communicate can also add files (e.g., /proc/pci on PCI bus computers). There are also files that are present or not, depending on which kernel options are compiled. (My /proc/modules is empty, because I did not compile in modules support.)

Some Common Files in /proc

- **cpuinfo** contains the information established by the kernel about the processor at boot time, e.g., the type of processor, including variant and features.
- **kcore** contains the entire RAM contents as seen by the kernel.
- **loadavg** contains the system load averages for the last 1, 5 and 15 minutes, along with the number of processes currently running and the total number of processes.
- **meminfo** contains information about the memory usage, how much of the available RAM and swap space are in use and how the kernel is using them.
- **stat** contains system statistics, counts of the amount of usage the kernel has made of basic system resources.
- **uptime** contains the amount of time in seconds that the system has been running, and the amount of that time that it has been idle.
- **version** contains the kernel version information that lists the version number, when it was compiled and who compiled it.

- **net/** is a directory containing network information.
- **net/dev** contains a list of the network devices that are compiled into the kernel. For each device there are statistics on the number of packets that have been transmitted and received.
- **net/route** contains the routing table that is used for routing packets on the network.
- **net/snmp** contains statistics on the higher levels of the network protocol.
- **self/** contains information about the current process. The contents are the same as those in the per-process information described below.
- ***pid*/** contains information about process number *pid*. The kernel maintains a directory containing process information for each process.
- ***pid*/cmdline** contains the command that was used to start the process (using null characters to separate arguments).
- ***pid*/cwd** contains a link to the current working directory of the process.
- ***pid*/environ** contains a list of the environment variables that the process has available.
- ***pid*/exe** contains a link to the program that is running in the process.
- ***pid*/fd/** is a directory containing a link to each of the files that the process has open.
- ***pid*/mem** contains the memory contents of the process.
- ***pid*/stat** contains process status information.
- ***pid*/statm** contains process memory usage information.

You can look at the contents of these files yourself. Just type:

```
cat /proc/meminfo
```

and you will see something like:

```
     total:  used:  free:  shared: buffers:
Mem: 11423744 8753152 2670592 2670592 2764800
Swap: 25800704 5328896 20471808
```

This table shows you how much memory you have, the amount you are using and how it is being used.

Of What Use Is All This Information?

Most people are using /proc without realizing it. The programs **top**, **ps**, **free** and their friends all use /proc. The information that they provide is taken directly from /proc and formatted for display.

Compare the contents of the /proc/meminfo file (above) with the command **free** that gives output that looks like:

```
        total    used    free   shared  buffers
Mem:     11156    8680    2476    2724    2800
Swap:    25196    5204   19992
```

As you can see, this table is just a formatted version of the contents of /proc/meminfo.

The output of the **ps** program is all available in /proc; all of the information is stored in the per-process directories. Most of it just needs to be reasonably formatted for the user.

What is ProcMeter?

ProcMeter is a program that monitors the information stored in /proc. The information is displayed in a number of graphs. Each of these graphs shows one aspect of the system. The program runs under X Windows on Linux only.

Anybody who has used **xload**, **xmeter** or **perfmeter** will recognize this description. The difference is these programs use a system-independent method of obtaining data, whereas ProcMeter was designed for Linux from the start. When ProcMeter is using /proc, it is occupying minimal memory and taking negligible CPU time. Once /proc is used, other ideas for obtaining data spring to mind. Looking at the table of /proc above, we can see there is a lot of useful information available.

What Can ProcMeter Tell Me?

The statistics available in ProcMeter can be divided naturally into a number of classes.

1. Processes—Basic information about the system, how busy it is and how heavily loaded it is. The processing power of the CPU is spread between all of the running processes and the kernel, and is idle for the remaining time.

cpu	is the total percentage of the CPU being used.
cpu-user	is the percentage of the CPU used by user processes.
cpu-nice	is the percentage of the CPU used by nice (low priority) processes.
cpu-sys	is the percentage of the CPU used by the kernel.

cpu-idle	is the percentage of the CPU unused (opposite of CPU).
load	is the system load, the number of running processes averaged over the previous minute.
proc	is the number of processes present on the system.
context	is the number of context switches between processes per second.

1. Memory (Real and Virtual)--Memory is such a precious resource (especially on small PCs for the home user) that it is important to keep track of it. The beauty of the Unix system is that the use of virtual memory (swap space) is transparent. Transparent, that is, until your computer makes a noise like a coffee grinder, and programs start to crawl—this is a sure sign that you are out of real memory and living in the virtual stuff.

mem-free	is the amount of free RAM.
mem-used	is the amount of used RAM.
mem-buff	is the amount of RAM used for file buffers.
mem-cache	is the amount of RAM used as cache memory (kernel version 2.0).
mem-swap	is the amount of swap space on disk being used (the shortfall in RAM).
swap	is the amount of swapping (the sum of swap-in and swap-out).
swap-in	is the number of pages of memory swapped in from disk per second.
swap-out	is the number of pages of memory swapped out to disk per second.

1. Hardware—The hardware the operating system runs on is often a bottleneck in performance. Every interrupt that is generated by hardware

must be processed by the kernel. The disk drive, another slow device, must also be controlled.

page	is the amount of paging (the sum of page-in and page-out).
page-in	is the number of pages of memory read in from disk per second.
page-out	is the number of pages of memory written out to disk per second.
disk	is the number of disk accesses per second.
intr	is the number of interrupts (IRQs) per second.

1. Network—When running on a network, there can be a quite an impact on system performance due to handling the traffic. Each packet that arrives must be handled promptly, causing hardware interrupts and kernel CPU usage.

lpkt	is the number of packets on local interfaces (same machine).
fpkt	is the total number of packets on fast network devices.
fpkt-rx	is the number of received packets on fast network devices.
fpkt-tx	is the number of transmitted packets on fast network devices.
spkt	is the total number of packets on slow network devices.
spkt-rx	is the number of received packets on slow network devices.
spkt-tx	is the number of transmitted packets on slow network devices.

How Can ProcMeter Help?

I have a ProcMeter window permanently open occupying the right-hand edge of the screen. Most of the time it is just taking up space, but there are times when it can be very useful.

When getting files via FTP from the Internet, the amount of packets sent and received can be monitored. When the packets stop coming, the transfer is finished or stuck. This is a good time to ping the FTP site to see if the route is still open.

Have you ever felt that the program you have just written is taking too long to run? This could be a symptom of running out of RAM and using swap space. Try looking at the mem-used, mem-free, mem-swap and swap graphs. A steeply rising graph will indicate a memory leak.

Where to Get ProcMeter

The latest version of ProcMeter is version 2.1, available as source code by FTP from sunsite.unc.edu. The file is called procmeter-2.1.tgz, and is in the directory /pub/Linux/X11/xutils/status. Alternatively, if you have WWW access, the latest information about ProcMeter is available on my home page at http://www.gedanken.demon.co.uk/, along with links to other sources for the program.

Figure 1

Figure 2

Andrew Bishop has been using Linux for 2 years. The original version of ProcMeter was the first program he wrote using Linux. He programs mainly in C, Perl and Emacs Lisp on Unix systems, often inventing his own version of the wheel as he goes. He can be reached by e-mail at ambi@gedanken.demon.co.uk.

Archive Index Issue Table of Contents

Advanced search

Somebody Still Uses Assembly Language?

Richard Sevenich

Issue #36, April 1997

Assembly language is a wonderful tool for teaching about how computers work. Professor Sevenich explains how it is used at WSU.

In the core program for our computer science curricula we offer two assembly language courses as elements in that part of our sequence providing hardware emphasis. Although the students do learn to program in this arcane language, the emphasis is on using assembly language as a detective's tool to learn about the underlying hardware.

Both courses involve the omnipresent Intel 80*x*86 architecture. However, the first course treats the chip as an 8086/88 and works within the MS-DOS environment. Insofar as is practical within the existing time constraints, we pretend that MS-DOS is not present and try to simulate an embedded systems environment. The essential fact is that MS-DOS puts us in charge of the system resources, i.e., in real address mode. This first course is a prerequisite for our subsequent hardware courses.

The focus of the second course is to examine the architecture elements that support a multitasking, multiuser operating system. For this course we have chosen Linux as the environment. This second course is a prerequisite for subsequent hardware courses as well as for our operating systems sequence. Linux is typically used in the latter. Of course, in such a sophisticated multitasking, multiuser system we no longer have direct control over the hardware resources. It is of central interest to see how the operating system protects itself.

This article discusses the second course. The intended audience consists of those who have an interest in the features of the 80*x*86 (*x* >= 3) Intel

architecture that support an operating system such as Linux. The two techniques we use for investigating are as follows:

1. Write our own assembly language code to probe the architecture.
2. Examine assembly language written by others.

These two approaches are discussed in their respective sections later in this article. This article is not an attempt to investigate the Intel architecture (a subject for a large volume), but to describe the tools and resources available to do so.

Virtually all textbooks on the Intel 80*x86* architecture assume that the reader is working in a Microsoft environment, usually with the Microsoft Assembler, MASM. Because we are working in a Linux environment, we do not use such traditional textbooks; instead we use as the primary resource the *Intel486 Processor Family: Programmer's Reference Manual* (1995), Intel Order Number 240486-003. This is a large manual and of special interest are Parts I and II dealing with application and system programming, respectively. Other useful resources are the on-line *Kernel Hacker's Guide* (see http://www.ssc.com/linux/ldp.html), *Brennan's Guide to Inline Assembly* (see http://www.rt66.com/-brennan/djgpp/djgpp_asm.html), and the various man pages and info documents available within Linux itself. Using such a set of resources rather than a focused textbook is, of course, typically how a *real world* software engineer operates.

Why Use Linux? Which Distribution?

Linux is a natural choice for rather obvious reasons:

1. It is free.
2. It includes a complete set of development and detective tools.
3. The source code is available.
4. It is an evolving multitasking, multiuser environment making use of the advanced features of the underlying chip architecture.

I recommend Debian GNU/Linux to our students because:

1. It is quite stable.
2. It can be updated/upgraded nondestructively, in place.
3. Various libraries are in the standard locations.
4. It is non-commercial, so students can get more seriously involved with maintenance and development later in the curriculum.
5. The Debian users and developers are extremely responsive and helpful.

Other distributions such as Red Hat or Craftworks meet most of these requirements quite well also, except for item 4, which is important for our students, but perhaps not to others.

Writing Our Own Assembly Language Programs

We have found it convenient and productive to write our assembly language in-line within C source code. Labels can be interjected in the source code at appropriate places to provide breakpoints for the debugger. The primary motivation for writing in-line assembly language is to examine architectural features. The assembly language statements are AT&T style rather than Intel style. The former seems to be the Unix custom.

As a simple example, we'll exhibit a short program, example1.c (see Listing 1), whose purpose is to examine the flags register which has three types of flag bits: status bits (e.g., the Carry Flag), system flags (e.g., the two bit combination giving the I/O Privilege Level), and a control flag, the Direction Flag. The program does the following:

1. Puts a copy of the flags register in the eax register for examination (breakpoint bp1).
2. Flips all the bits in that copy (breakpoint bp2).
3. Attempts to write that bit-flipped copy into the flags register and then puts a copy of the resulting flags register into eax for examination (breakpoint bp3).

Note how in-line assembly language is supported by the **asm** macro.

To compile this into the executable program example1.x, containing necessary information for subsequent use by the debugger, we use the **-g** switch in the following command:

```
gcc -g example1.c -o example1.x
```

The next step is to invoke the debugger. It is convenient to also get a log of the debugger activities via a pipe to the tee command so the command line entry would be:

```
gdb -silent example1.x | tee example1.log
```

yielding the gdb prompt

```
(gdb)
```

Now gdb is ready to run example1.c, while tee will produce a record of our activity in example1.log. The latter can be printed or examined with an editor.

It is beyond the scope of this article to also be a tutorial on the use of gdb; such documentation is readily available in man page and info format. In addition, for use within a browser, one can find, in html format, the FSF document *Debugging with gdb* by Stallman and Pesch. One current URL for this is: http://funnelweb.utcc.utk/~harp/gnu/tars.

It might be more efficient to first look at the terse, readable introduction to gdb given in *Getting to Know gdb* by Loukides and Oram in Issue 29 of *Linux Journal* (September 1996).

Having said that, let's at least show a typical example1.log (see Listing 2) which shows setting breakpoints and then stopping at those breakpoints to examine registers of interest. Lines starting with the **(gdb)** prompt are commands entered by the user, whereas everything else is information volunteered by the debugger.

The log file tells the following:

- The original value of the flags register was 0x246.
- `Our attempt to flip all the bits and write the flipped value back to the flags register was only partially successful and that attempt generated an exception (signal **SIGTRAP**).

The investigator might go through a questioning process rather like this:

- What does the original value of the flags register mean in terms of individual bits (e.g., what is the I/O Privilege Level)?
- Which instruction generated an exception and why?
- Which bits could be flipped and which could not? Why?

Interesting facts are then uncovered. For example, in the log file shown, the ID flag (bit 21) was successfully flipped. According to the Intel documentation this indicates that this processor can execute the CPU_ID instruction. On the other hand, the bits giving the I/O Privilege Level (bits 12 and 13) could not be modified. Clearly, that is expected—the casual user should not be able to change anything that might help get at the I/O hardware directly.

Examining Assembly Language as Written by Others

Typically, even for device drivers, Linux developers do not use assembly language. Hence, it is particularly revealing to examine those very few parts of the kernel which *are* written in assembly language. These can be found within the Linux distributions with the command:

```
find -name *.S
```

entered from the root directory. Of particular interest are these:

- bootsect.S (Intel style instructions)
- setup.S (Intel style)
- head.S (AT&T style)

These are heavily commented, but additional guidance can be found in the Intel documentation and in Alessandro Rubini's *Tour of the Linux Kernel Source*, found in the Kernel Hacker's Guide. These modules do the first portions of system initialization, a process which is completed by C routines. Once they have been executed, the assembly language routines are done. Another module of interest is entry.S (AT&T style) whose tasks are ongoing. In particular, it contains low level routines for handling system calls and faults.

Conclusion

This material should help interested readers start their own investigations of the Intel 80*x*86 (*x* >= 3) architecture and the Linux kernel. Much can then be learned about such topics as operating modes, memory management, and building the various descriptor tables.

Richard A. Sevenich is a Professor of Computer Science at Eastern Washington University in Cheney, Washington. His original enthusiasm for Linux was derived in part from the fact that its development had been driven by user needs rather than by marketing hype. He can be reached at rsevenich@ewu.edu.

Archive Index Issue Table of Contents

Advanced search

Applixware

Gary Moore

Issue #36, April 1997

If you install Applixware on your system, you'll notice an impact on system resources.

- Product: Applixware 4.2 For Linux
- Publisher: Red Hat Software, Inc.
- Phone: 800 454-5502
- Fax: 203 454-2582
- WWW: http://www.redhat.com/
- Price: USD $495, student price USD $79.95
- Reviewer: Gary Moore

An Applixware On-Line Book

Applixware is is an excellent “office suite” that may open doors to wider use of Linux.

Applix Words

Applixware features a word processor, a spreadsheet, a presentation graphics tool, a drawing tool, an e-mail client, database connectivity and an object-oriented application builder. For some time, this professional set of programs has been available for other Unix platforms, including HP-UX, Solaris, AIX and Digital Unix, and now Applixware is available for Intel-compatible Linux machines and Microsoft Windows; at the time of this writing, the NT version is out and the 95 version is in beta testing.

If you install Applixware on your system, you'll notice an impact on system resources. A complete installation with the included Red Hat RPM files requires 210MB—if that's more than you have available, you can make a partial installation from a live, “unpacked” directory on the CD-ROM. In fact,

Applixware can be launched and used directly from the CD-ROM, though this makes program operation a little leisurely. I was using Red Hat Linux 4.0 when I reviewed Applixware, but the software should work fine on other distributions, and installation instructions are included.

The CD speed may not seem bad if you're using a 486DX25, on which Applixware is fast enough to be usable, but probably too slow for a production environment; I found my meager CPU power to be a real problem only when I started using the graphics tools.

This is not an application for low-memory systems. As cheap as RAM is today, this shouldn't be too painful a state to rectify. With 16MB of RAM, the word processor was snappy enough with X and the Afterstep window manager running, but having much else loaded caused so much paging of virtual memory I needed something to read while waiting.

Not much reading material comes with Applix—at least, not on paper. Back when it was known as Asterix and also in version 3.x of Applix, there was a manual for each module, but either with the Linux version or with the later releases, virtually all documentation is in the “On-Line Books”. Use the on-line tutorials if you're new to the system, or the on-line help if you just need a reference.

Applix Graphics

Applix Words is a full-featured word processor with everything you'd expect to find in a modern product. That is, unless you're looking to do something which really should be done using desktop publishing software. By the way, one thing you never want to do with it is embed, oh, 80 or so large, 256-color GIFs in a single document—at somewhere around 8MB, application behavior gets a bit wacky. Linking is much, much better.

Words gives you tables, borders, shading, embedded equations and calculations, conditional text and cross-referencing, international dictionaries, thesauri and a multi-font, multi-size WYSIWYG display. You can rely on multiple undo and redo, and when you're done, you can save PostScript and PCL printer files or send them directly to a networked printer.

HTML is easy with the Applix HTML authoring tool. Documents can be imported from Applix or another popular word processor using one of the format filters or created from scratch with the same ease as a word processing document. Clip art, GIFs and linked or embedded Applix Graphics images are converted seamlessly. *Applix Spreadsheets* documents and queries from the database

interface application, *Data*, can be included, too. Tables, colors, and more than 25 standard HTML styles are all under your control.

Applix Builder

Applix Graphics is a terrific drawing and presentation graphics tool. At your disposal are user-definable fill patterns, various brush styles, shearing, drop shadows, incremental zoom, rotating, scaling, color pixel editing and text wrapping, to name a few. Grid snap, guide lines, rulers, and coordinates help create precise and complex drawings quickly. I found graphics as easy to produce with *Graphics* as with *Powerpoint*.

The good news continues with *Applix Spreadsheets* with calculation-based attributes, 3D charts, named views and dynamic links to objects in other Applixware applications. When your linked data from elsewhere changes, it is automatically updated in your spreadsheet. There are live links to a relational database through *Applix Data*, goal seeking, drag-and-drop, projection tables and background recalculation. You can import those old Lotus 1-2-3 and Excel spreadsheets, too.

You might not think you need another mail client, but check out *Applix Mail*. When you receive mail, a dialog box pops up with the sender name and subject, giving you the options “Read Now”, “Read Later”, and “Help”. You can attach Applix files to your mail messages and upon receipt, launch the appropriate Applix tool for viewing. Mail can be marked “Urgent”, marked with a “Reply by” date, and also sent by “certified” mail, giving you a receipt when the recipient has read the mail. Of course you can “Cc” and “Bcc” people. You also get shared mail folders, automatic conversion of messages and documents to your preferences, encryption, and mail filtering based on rules you specify.

Applix Data connects Applixware applications to SQL databases like Informix, Oracle, Ingres, and Sybase, seamlessly querying data from one or more tables, selecting information with query conditions, and performing advanced queries and joins. Rows can be edited, inserted, and deleted. A live link in your document to the database means up-to-date data. *Data* provides a lot of capability when teamed with ELF and *Builder*.

The Extension Language Facility (ELF) is an interpreted language with which users can build and deploy applications, front-ends to applications, automate tasks and connect to databases and other external sources of data. The Applix user interfaces are built with ELF and ELF macros can be used to automate tasks in any of the Applixware applications. Some capabilities include: TCP/IP socket interfacing, remote procedure calls, interactive debugging, many built-in macros, string manipulation, and arithmetic and Boolean operators.

Builder is object oriented and gives you access to external data sources as well as the capabilities of the Applixware application suite for use in your custom applications. Also, full access to ELF macros and functions, external objects, shared classes, RPC and shared library support. Plus, the applications you develop in *Builder* on one platform are portable to Applixware on other platforms without modification.

Applixware is a terrific package. When I heard it was available for Linux, I knew I could let go of Microsoft Office (and MS Windows) forever.

Gary Moore is the Editor of *Linux Journal*.

Archive Index Issue Table of Contents

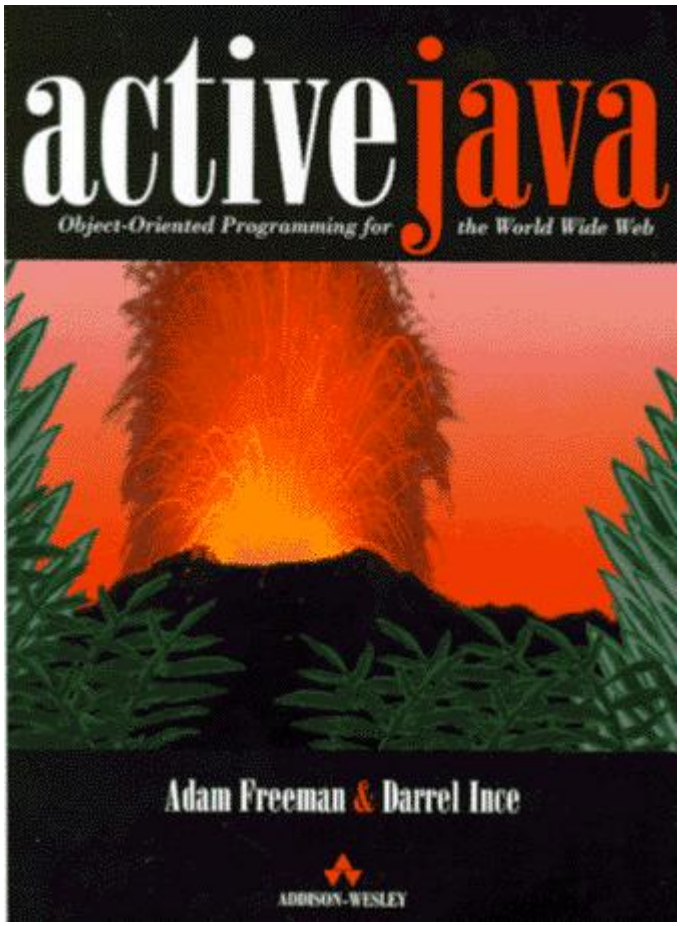
Advanced search

Book Reviews: Active Java and Exploring Java

Danny Yee

Issue #36, April 1997

The book works its way through the elements of the language, explains how to use the awt and net libraries, introduces the Java Development Kit and the basics of writing applets and applications—and then concludes with a chapter on Java internals.



- Title: Active Java: Object-Oriented Programming for the World Wide Web
- Author: Adam Freeman & Darrel Ince
- Publisher: Addison-Wesley

- Price: US$25.95
- ISBN: 0-201-40370-6
- Title: Exploring Java
- Author: Patrick Niemeyer & Joshua Peck
- Publisher: O'Reilly & Associates
- Price: US$24.95
- ISBN: 1-56592-184-4
- Reviewer: Danny Yee

Somewhat bemused by the marketing frenzy and in no particular hurry to learn yet another programming language, I have refrained from asking for review copies of any books on Java. Nevertheless, a few turned up on my doorstep anyway, and I found it hard to resist finding out what all the fuss is about.

The first book to arrive, and the only one I read right through, was *Active Java*. This is an introduction to Java aimed at those having basic programming competence but no experience with an object-oriented language. The book works its way through the elements of the language, explains how to use the awt and net libraries, introduces the Java Development Kit and the basics of writing applets and applications—and then concludes with a chapter on Java internals. The emphasis is on covering important ideas and concepts rather than on providing details. *Active Java* is easy to follow and clearly laid out, and I recommend it for anyone wanting a broad overview of Java. I think it would also make a good textbook for an undergraduate course, though it lacks exercises and is perhaps not repetitive enough.

As a supplement to *Active Java*, and a source of more detailed information, I used *Exploring Java*. This begins with a brief look at internals and security issues and then launches into a basic “Hello Web!” applet. This book contains detailed descriptions of the basic classes and standard libraries and is clearly aimed at experienced programmers who want to learn Java in order to write serious applications.

I have only glanced at the three other books on Java that arrived; *Java in a Nutshell* (O'Reilly) looks like a reference for the serious Java programmer; *On To Java* (Addison-Wesley) is a textbook with an unusual layout, using paragraphs numbered sequentially throughout; and *Learn Java on the Macintosh* (Addison-Wesley) comes with a Mac version of the Java Development Kit on CD-ROM. Anyone looking for a book on Java should search carefully: as even this small sample illustrates, there are books on Java for all sorts of niche markets. I wouldn't be at all surprised to see titles like *From Common Lisp To Java For Amiga Users* and *101 Implementations of Tetris In Java* appearing!

Danny Yee received review copies of the books mentioned from Addison-Wesley and O'Reilly & Associates, but has no stake, financial or otherwise, in their success. He can be reached at danny@cs.su.oz.au.

Archive Index Issue Table of Contents

Advanced search

Using Perl to Check Web Links

Jim Weirich

Issue #36, April 1997

Do you have many links on your web pages? If so, you're probably finding that the pages at the ends of those links are disappearing faster than you can track them. Ah, but now you can get the computer to handle that for you.

One of the first things I did when I got my first Internet account was put together my own set of web pages. The one I get the most comments about is called “Weirichs on the Web” where I link to other Weirichs I have found on the Web. Although a lot of fun, keeping the links up to date can be very tedious. As web pages that I reference are moved or deleted, links to them become stale. Without constant checking, it is difficult to keep my links current.

So, I began to wonder, is there a way to automatically find the outdated links in a web page? What I needed was a script that would scan all of my web pages and report every bad HTML link along with the web page on which it was used.

There are several parts to this problem. Our script must be able to:

- fetch a web document from the Web
- extract a list of URLs from a web document
- test a URL to see if it is valid

The LWP Library

We could write code by hand to extract URLs and validate them, but there is a much easier way. LWP is a Perl library (available from any CPAN archive site) designed to make accessing the World Wide Web very easy in Perl. LWP uses Perl objects to provide Web-related services to a client. Perl objects are a recent addition to the Perl language and many people might not be familiar with them.

Perl objects are references to “things” that know what class they belong to. These “things” are usually anonymous hashes but *you don't need to know this*

to use an object. Classes are packages that provide the methods the object uses to implement its behavior. And finally, a method is a function (in the class package) that expects an object reference (or sometimes a package name) as its first argument.

If this sounds confusing, don't worry. Using objects is very easy. LWP defines a class called **HTTP::Request** that represents a request to be sent on the Web. The request to ***GET*** a document at URL http://w3.one.net/~jweirich can be created with the statement:

```
$req = new HTTP::Request GET,
 'http://w3.one.net/~jweirich';
```

new creates a new Request object initialized with the ***GET*** and http://w3.one.net/~jweirich parameters. This new object is assigned to the **$req** variable.

Calling a member function of an object is equally straightforward. For example, if you want to examine the URL for this request, you can invoke the **url** method on this object.

```
print "The URL of this request is:
", $req->url, ",\n";
```

Notice that methods are invoked using the **->** syntax. C++ programmers should feel comfortable with this.

Getting a Document

All the knowledge about fetching a document across the Web is stored in a UserAgent object. The UserAgent object knows how long to wait for responses, how to handle errors, and what to do with the document when it arrives. It does all the hard work—we just need to give it the right information so that it can do its job.

```
use LWP::UserAgent;
use HTTP::Request;
$agent = new LWP::UserAgent;
$req = new HTTP::Request ('GET',
 'http://w3.one.net/~jweirich/');
$agent->request ($req, \&callback);
```

This snippet of Perl code creates a UserAgent and a Request object. The Request method of UserAgent issues the request and calls a subroutine called **callback** with a chunk of data from the arriving document. The **callback** subroutine may be called many times until the complete document has been received.

Parsing the Document

We could use regular expressions to parse the incoming document to determine the location of all the links, but when you begin to consider that HTML tags may be broken across several lines and all the little variations involved, it becomes a more difficult task. Fortunately, there is an HTML parsing object available in the LWP library, called **HTML::LinkExtor**, which extracts all the links from an HTML document.

The parser is created and then fed pieces of the document until it reaches the end of the document. Whenever the parser detects links buried in HTML tags, it calls another callback subroutine that we provide. Here is an example that extracts and prints all the links in a document.

```
use HTML::LinkExtor
$parser = new HTML::LinkExtor (\&LinkCallback);
$parser->parse ($chunk);
$parser->parse ($chunk);
$parser->parse ($chunk);
$parser->eof;
sub LinkCallback {
    my ($tag, %links) = @_;
    print join ("\n", values %links), "\n";
}
```

Putting It Together

We now have all the tools we need to build our **checklinks** script. We will define two operations for URLs. When we scan a URL, we will fetch the document (using a UserAgent) and scan it for internal HTML links. Every new link we find will be added to a list of URLs to be checked.

Next, check a link to see if it points to a valid web document. We could try retrieving the entire document to see if the document exists, but the HTTP protocol defines a ***HEAD*** request that gets only the document's date, length and a few other attributes. Since a ***HEAD*** request can be much faster than a full ***GET*** for large documents, and since it tells us what we need to know, we will use the **head()** function of the **LWP::Simple** package to check a URL. If **head()** returns an undefined value, then the document specified by the URL cannot be fetched and we add the URL to a list of bad URLs. If **head()** returns a list, the URL is valid and it is added to the list of good URLs. Finally, if the valid URL points to a page in our local web space and ends with ".html" or ".htm", we add the URL to a list of URLs to be scanned.

The scanning process produces more URLs to be checked. Checking these URLS produces more URLs that need to be scanned. As URLs are checked, they are moved to the good or bad list. Since we restrict scanning to URLs in our local web space, eventually we will scan all local URLs that are reachable from our starting document.

When there are no more URLs to be scanned and all URLs have been checked, we can print the list of bad URLs and the list of files that contain them.

Results

The complete code to **checklinks** is found in Listing 1. You will need Perl 5 to be able to run the **checklinks** routine. You will also need a recent copy of the LWP library. When I installed LWP, I also had to update the IO and Net modules. You can find Perl, and the LWP, IO and Net modules at http://www.perl.com/perl.

You can invoke **checklinks** on a single URL with the command:

```
checklinks url
```

If you wish to scan all local URLs reachable from the main URL, add a **-r** option.

Running checklinks on my home system against my entire set of web pages took about 13 minutes to complete. Most of that time was spent waiting for the bad URLs to timeout. It scanned 76 pages, checked 289 URLs, and found 31 links that were no longer valid. Now all I have to do is find the time to clean up my web pages!

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 http://w3.one.net/~jweirich.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Creating a Multiple Choice Quiz System with CGI

Reuven M. Lerner

Issue #36, April 1997

Designing quizzes the easy way using CGI.

Over the last few months, we have looked at a number of techniques that CGI programmers can use to work on their programs. This month, we will look at a multiple-choice quiz system that uses a combination of techniques to create a simple, but effective, system for creating quizzes for our users. By the end of this short project, you will have not only a good idea of how to implement this type of interaction, but a working four-question quiz, as well.

Before we can begin, we will need to decide on a file format which will contain the questions and answers for our quiz. We could put all of the questions and answers inside of the program itself, but moving them to one or more external files will let us reuse the software with other quizzes on our system. Given that this is a simple quiz, let's say that the questions and answers for each quiz are stored in a file whose name is the same as the quiz name. Thus the quiz named “presidents” will be stored in a file named “presidents”, while the quiz named “unix” is stored in a file named “unix”.

Now that we have decided on filenames, we need to decide on a format for the contents of the file. Let's take the easy route, and put one question and its associated possible answers on each line in the file, each separated by tabs, and ending with the letter “a”, “b”, “c” or “d”, that corresponds to the correct answer.

So that the file can contain comments and whitespace, we'll say that any line beginning with a hash mark (#) is considered a comment, to be ignored. The same goes for any line consisting solely of whitespace. Allowing for comments and whitespace makes it possible for us to comment out questions that we no longer want to use, without having to delete them altogether.

Here is a sample quiz on the subject of cranberries, which we will put in a file named, oddly enough, “cranberries”:

```
# This is the quiz file about cranberries.
```

```
# Comment lines contain a hash mark (#) in the
# first column, and are ignored, as are lines
# containing only whitespace.
```

```
What color are cranberries?     Red     White\
        Blue    Dark green      A
What can you make with cranberries?     Muffins\
        Sauce   Steak   A and B D
```

Note that the questions and answers in this file can contain space characters, but *not* tab characters. This will typically not affect things very much, but it is a consideration to keep in mind. Also, while each line can be as long as needed, the question and its associated answers **must** remain on a single line of text (that is, must end in a carriage return).

Creating an Object

Our quiz program actually consists of two different programs working in concert with each other. The first, **askquestion.pl**, produces an HTML form that presents the user with a question and a list of possible answers. That form will be submitted to another CGI program, **checkanswer.pl**, which determines whether the user has selected the correct answer.

Because both of these CGI programs will have to access the same quiz file, it is probably a good idea to centralize such functions in a single Perl 5 object. Such an object would have to read the file and return a question of our choosing from the list of available questions. To make things a bit more interesting, this object should include a method that retrieves a random question from the file, which makes the quiz less predictable for the user.

The object that we will use in our quiz program is shown in Listing 1. All this code means is that you can place a:

```
use QuizQuestions;
```

statement near the top of both CGI programs to create a Perl object that reads the questions to the “cranberries” quiz. To do this, you can use this statement:

```
my $quiz = new QuizQuestions("cranberries");
```

For example, you could retrieve the fifth question with:

```
my @question = $quiz->getQuestion(5);
```

or a random question with:

```
my @question = $quiz->getRandomQuestion;
```

As you can see, the **QuizQuestion** object in Listing 1 has nothing to do with CGI programming per se. Even if we were creating a quiz system that wouldn't be used on the Web, this object would be a good starting point. By using an object to represent our data, we have also made it possible to change the file format we are using without modifying the CGI programs that access the data. If we were so inclined, we would be able to move the quiz data into an SQL table, and access it via a database client from within Perl. As long as the interface to the outside world remains the same, our CGI programs wouldn't care.

Asking the Right Questions

Now that we have created a fairly simple interface to the quiz data, let's create the first of our two programs, **askquestion.pl**. This program produces an HTML form which not only asks a question, but also lets the user choose an answer by clicking on the appropriate radio button.

One possible version of the program, shown in Listing 2, is fairly straightforward. It creates one instance of CGI, an object which helps us write CGI programs, and one instance of QuizQuestions, the object we created above. After instantiating these two objects, we then produce a simple HTML form containing four radio buttons that correspond to each possible answer. We then create a submit button and a reset button, and finish creating the HTML form.

However, we also create a hidden field that contains the number of the question the user is answering. This number is returned by the **getQuestion** and **getRandomQuestion** methods within QuizQuestions. If you didn't understand previously why we needed to return these values along with the questions and answers, perhaps it will be clearer now. HTTP is a stateless protocol—every request made to a server is independent of any other requests made to it. The quiz requires at least two connections to the HTTP server—one to get the question and produce a form with **askquestion.pl**, and a second to submit the user's response and check the answer, **checkanswer.pl**.

The problem is that **checkanswer.pl** can verify only that the user's answer is correct if it knows which question the user was asked. Since **checkanswer.pl** is invoked with a separate request to the HTTP server it cannot know which question was selected, unless we have some way of passing that message from the invocation **askquestion.pl**.

We could have used the hidden field to pass the correct answer along to checkanswer.pl, but this is a bad idea because hidden fields are hidden only from obvious view. If a user were interested in finding the correct answer, he or

she would be able to look at the page's HTML source, which would quickly reveal the answer. This way, users know only which question is being asked, not which answer is correct.

Also note that the name of the quiz comes from the query string, which is passed to us in the **QUERY_STRING** environment variable. This lets us, as mentioned above, use the same quiz program for multiple programs. By changing the value placed in the query string, you can turn this pair of programs into many different quizzes, each with its own set of questions and answers. When we set the **action** attribute in the **<Form>** tag, we make sure that it includes not only the name of the program to which the form should be submitted, **checkanswer.pl**, but also the name of the quiz, which appears in the query string.

Ending the Suspense

As we saw earlier, the form generated by **askquestion.pl** is submitted to a second CGI program, **checkanswer.pl**. Checkanswer.pl opens the list of questions, retrieves the question that the user was asked by retrieving the value of the **questionNumber** form element, which is hidden in the form, and checks the user's answer against the correct one.

If the user answers the question correctly, the program displays a "congratulations" headline along with the correct answer, and asks if the user would like another question.

If the user answers the question incorrectly, the program displays the correct answer, offers some consolation, and asks the user if he or she would like to continue.

Now you can see why you need the getQuestion and the getRandomQuestion method. With getQuestion alone, you can retrieve a question, but not a random selection from the list of questions. But if you had only getRandomQuestion, you would not be able to retrieve the question that the user had asked, and thus would not be able to check the user's answer against the correct one.

The source code for **checkanswer.pl** is in Listing 3. One obvious flaw of this implementation is that if the site administrator decides to modify the questions file between the time the user receives the question and when he or she submits the form, the question might be marked as wrong. That's because the programs expect the order of the questions will not be modified between the time the question is asked and when it is answered. If you were to insert a new question at the top of the file, this would turn question 1 into question 2, question 2 into question 3 and so on—which would mean that checkanswer.pl would compare the user's answer with an answer to a different question.

Note that we used Perl's **eval** function to get the actual text of the answer. Perhaps this is simply a personal hang-up, but I hate it when I am told that I answered incorrectly, but no one tells me what the correct answer was. We could have stored the answers in an associative array, but I decided that it would be interesting to use **eval** to get the value of a variable. In this case, we concatenate the string "$answer" and the value of $rightAnswer, giving us one of the four possible strings "$answerA", "$answerB", "$answerC" or "$answerD". **eval** is handed that string and returns the value of the variable named in the string.

The Initial HTML

Now that we have defined QuizQuestions, askquestion.pl and checkanswer.pl, all that remains is to create an HTML file that acts as the initial entrance into the quiz.

```
<HTML>
<Head>
<Title>Play our quiz!</Title>
</Head>
<Body>
<H1>Play our quiz!</H1>
<P>You can play our cranberry quiz by clicking
<a href="/cgi-bin/askquestion.pl?cranberries">
here</a>.</P>
</Body>
</HTML>
```

Notice that the URL leading to the initial question must have a quiz name appended to it in the query string. Other than that, though, this is a simple HTML document.

This quiz appears to work pretty well so far, although there are certainly features that you might add—such as a scoreboard, better error-checking when reading the quiz file, or a system that ensures that users don't see the same question twice.

But more important than any of these is the fact that while the format of the question file is easy for programmers to understand, non-programmers who would like to add, delete or modify questions might find the format confusing. Next month we will work on making this system more author-friendly, so that non-programmers can modify entries in the question file via an HTML form.

Reuven M. Lerner has been playing with the Web since early 1993, when it seemed 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

Letters to the Editor

Various

Issue #36, April 1997

Readers sound off.

The Global Perspective

I'm glad to extend my subscription because *Linux Journal* is a very nice Linux-related magazine providing me with the global perspective of the Linux world which cannot be easily obtained by searching the Web, Usenet, and the like. I am always grateful for your efforts and hope *LJ* will continue to help me as it did in the past year of my subscription.Sohn, Jung-woo Aerospace Engineering Department Seoul National University, Korea. sylph@plaza.snu.ac.kr

Is Java a Threat?

First, I would like to thank everyone at SSC for publishing *Linux Journal*. I eagerly wait for it to arrive at my door each month. I have been using Linux for a few years now and currently develope applications and utilities for Linux using Motif and, more recently, Java.

The concern I have for Linux is the ever-growing use of Java as a programming language of choice. I am not saying using Java to deploy applications on Linux is a bad thing, and I am not putting down the Java programming language (I use it regularly, and enjoy it). It just takes away some of the developer support for Linux-specific applications, which Linux needs. If applications are written in Java and run on Linux or any other operating system for that matter are we just turning our beloved operating systems into another Sun JavaStation or Oracle NC? Running Java-based applications on Linux is fine, but is Linux then losing the free and commercial application base that it needs to bring Linux to the next step? The next step being broad-based commercial Linux applications, that could be purchased at your local computer store or out of a general PC magazine. As many of us know, having a high number of applications available for an operating system attracts more people to it.

On the other hand, Linux has been a Java trail-blazer. Linux can incorporate Java binary support into its kernel. Creating applications in Java has its upside—applications which have been developed on a Microsoft platform can be easily run on the Linux operating system. Because of this, Linux could become a major platform in small- and medium-sized businesses that need both a rock solid operating system to run their operation and to be able to run all of the big name brand software. The Java language could actually increase the Linux-installed base around the world. If users of other operating systems become tired of software upgrade costs, downloading bug fixes and paying huge per minute costs to telephone support companies, maybe Linux will become the logical decision. Stover Babcock stover@pit-stop.com

The history of personal computers seems to have many examples of systems clutched too tightly by their creators, missing out on opportunities to broaden appeal and usage. Making Linux Java-capable does not change the essential nature of Linux: free and open.

Multithreaded Programming Library

In my article on multithreaded programming [“What is Multi-Threading?”, Issue 34], I completely failed to mention where to obtain the library I make use of in the programs that go with the article. The library I use came from http://pauillac.inria.fr/~xleroy/linuxthreads and is very good, and less than 100K to download. Cheers, Martin McCarthy marty@ehabitat.demon.co.uk

Finding a Users' Group

I just received my first issue of *Linux Journal* in the mail today. I must say that it is refreshing to know that there is a journal dedicated to Linux.

I live and play in Biloxi, Mississippi, an area not often thought of in positive, humanistic ways. But time and imported people have brought this area from gloom to glimmer. Technology has finally arrived in the deep south, sparked from casual conversations on a campus computer system, to a full blown ISP using Linux, of course. My Linux box is connected to the Internet via their service. Okay, so we have technology here, and we have people to utilize it. I know there is a small and quiet group of people out there who use Linux also. The gist of this letter is I would like to form a Linux users' group here, so the quiet group can join together and spread the wealth of knowledge out there with each other. For only by becoming a community of friends and associates, can we continue to grow and nurture our Love for Linux.Thank You Ted F. Fox tfox@gooner.datasync.com

This is a good opportunity to mention GLUE—Groups of Linux Users Everywhere. SSC, publishers of Linux Journal have established GLUE to give

users groups visibility, special deals on SSC products, and other services to help groups grow. See the home page (http://www.ssc.com/glue/) for details.

In Stores Now!

Greetings! I am writing in response to Cory Plock's letter to the Editor in the February 1997 issue of *Linux Journal* about not finding Linux distributions in stores.

I have been able to find Slackware and Red Hat in the major software stores in Houston for some time now, and I can even go into a few major book stores and not only find lots of books on Linux with their bundled distributions, but I have even found Red Hat 4.0 and Applixware on the shelf! I guess Linux has caught on earlier in some parts of the country than others. (At the first ever Houston Linux Users Group organizational meeting, we had over 150 people show up!) Don Harper Pencom Systems Incorporated duck@pencom.com

That's good news!

Archive Index Issue Table of Contents

Advanced search

Linux—The Internet Appliance?

Phil Hughes

Issue #36, April 1997

There has been talk about $500 appliances for connecting to the Internet. Were those people talking about Linux?

As the Internet grows it becomes a source for more and more useful but non-technical information. For example, the U.S. Postal Service has a web page that is very useful to anyone who wants to mail something. The Internet is also rapidly becoming the most cost-effective way to transfer information, since it offers much cheaper delivery than methods such as FAX machines.

It is in the best interest of companies using the Internet to distribute information for everyone to get on the Internet, so as to eliminate a duplicate channel for distributing information. While this won't happen overnight, some of them will be willing to help make it happen.

To this end, a $350 appliance that connects to your television has just been announced. However, it requires a TV, offers poor resolution and the input device is a glorified TV remote control. It could be used for some basic Internet access, but it isn't a serious approach to real Internet access.

What I want to do is encourage the Linux activists out there to think about proposing Linux as an important part of the solution. Linux is already a significant player on the web server end of the picture. Why not use Linux to build this “appliance” as well?

What Is Needed?

Before Linux can be the solution, we need to define the problem. From what I have read, this appliance is a low-cost computer system that can connect to the Internet, along with the necessary software to send and receive electronic mail and browse the Web. If you have been seriously working with Linux, you

probably think Linux already does this. Well, it does. That is, it does this job for you, but it is far from an appliance.

Think of a VCR—that's an appliance. You buy it, take it home, plug it into a couple of things, set the time and stations in your area, and you are up and running. Figure about 30 minutes to get it out of its box, connected and running.

Now, remember, lots of people think a VCR is too complicated. If you talk to VCR owners, you will probably find that quite a few have the time blinking 12:00, and that many don't know how to program their VCR to record a program at a time when they will be away from the TV.

Now that I have defined the average user, you should have a better idea of what is needed. In particular, something that is easier to set up than a VCR that will allow Internet access.

Can This Be Done?

Yes. And, with Linux? Yes again. In fact, it is more likely that it can be done with Linux than with other systems, because you have the source code to work with. And, as Linux is frugal with resources, it can be done with less investment in hardware.

Finally, if you do this, will it sell? Again, I feel the answer is yes. While many people need (or think they need) a computer system and will go for a low-end Mac or PC with MS Windows, there are many more people—millions—who don't want a computer, but do want access to information. Chances are that many of these people will later discover that they do want a word processor, a way to FAX and other computer-related tasks. If the cost of entry for the appliance is lower than a computer, lots of people will start with the appliance. With proper design, this appliance could also grow up into the other tasks while, of course, still running Linux.

How Do We Build It?

The key to success is to create something with high-functionality and a minimum cost. A basic 486 or 5x86 system with 16MB RAM and almost any disk should do it. It is probably worth including a CD-ROM drive in the package—2x units can be found for around $30 and this can really simplify things. Checking out the local computer shopper magazine proves that I could easily assemble these systems for $800 by buying everything at retail. And over $200 of this is the price of a monitor. I think this basic system could be produced for $500 in quantity—possibly less.

Now, this computer isn't good for much unless we can convert it from a general-purpose computer to an appliance that anyone can use. While not particularly hard, doing it correctly is the secret. Here are some considerations that help turn it into an appliance:

- Make it initially boot and load from CD.
- Use XDM so there is always a graphics screen.
- Have some built-in logins like mom and dad.
- Include support for all reasonable connectivity options including ISDN.
- Use diald so connections happen automatically.
- Cut a deal with some ISPs and have default connect files for them.
- On initial load, the system should ask for connection information. The questions should, whenever possible, be multiple choice.
- Include an automatic backup script for the configuration files.
- Include a decent web browser. Maybe Netscape would finally support Linux if this was done right.
- Build a web page where the users can get help/more information, and include this as the default location for the browser.

I think that will get things off to a good start. If a bunch of readers decide to give this a try, great. I think we could all learn from the experiences of others. Then, possibly, hardware manufacturers will realize there really is an opportunity for selling appliances here.

This Is Called Win-Win

There is a real chance for a serious win-win situation here. If all goes according to plan, there will be millions of new Linux boxes out there in the world, millions of people will get Internet access at a lower cost than they would have otherwise, some hardware manufacturers will make money, and some money will end up in the hands of people who worked on the software. For the long-term, these millions of new Linux users will soon want to buy that word processor, spreadsheet and personal information manager. Everybody wins.

Archive Index Issue Table of Contents

Advanced search

Usenix/Uselinux in Anaheim

Phil Hughes

Issue #36, April 1997

For those not familiar with Usenix, it has been the “wear a tie and get laughed at” Unix show for years.

Here I am at Usenix at the Mariott Hotel in Anaheim. Actually, it is pleasant to be in nice weather after almost drowning in Seattle. It had rained here the day before so the air was actually clean. But, let me talk about the show instead of the weather.

Usenix is a five-day show that, this year, has a heavy Linux presence. For those not familiar with Usenix, it has been the “wear a tie and get laughed at” Unix show for years. It is technical and tends to draw a very seriously technical crowd.

It is broken up into tutorials, a trade show and a technical conference. Well, plus the informal beer drinking sessions and such.

Tutorial Days

The first two days are tutorials and I elected to attend an all-day tutorial on the Linux 2.0 kernel presented by Stephen Tweedie. I found it to be excellent and that seemed to be the general opinion of the approximately 125 people who attended.

In eight hours and 170 overheads, Stephen addressed four specific areas of the kernel: memory management, the scheduler, filesystems and I/O and networking. I feel the goal of the talk, “to be with the design and algorithms behind the Linux kernel and to be able to read the Linux source code with some understanding” was met. While Stephen did not necessarily expect attendees to be familiar with Unix systems programming, the more you knew about Unix the easier it was to understand the presentation. After all, learning everything about a new operating system in eight hours is quite a challenge.

On Tuesday, Ted T'so gave a tutorial on writing device drivers under Linux. This talk was attended by about 60 students. I elected to take Tuesday as a day to catch up on *LJ* work and make a run to Fry's Electronics to see if they carry *Linux Journal*. They don't—which makes no sense as Fry's is exactly the kind of place a Linux geek would want to go.

Tuesday evening started with free food and drink. This is one of the best ways to get geeks talking. The Marriott did a great job with an array of food carts with various choices including fruit, veggies, potato patties, nachos, hamburgers and hot dogs. There were also drink and dessert carts. They even had my drug of choice, Dr. Pepper.

There were Birds-of-a-Feather sessions scheduled from 6PM to 10PM. The two Linux ones were scheduled at the same time, both at 7PM. As I already know a lot about Caldera Linux I elected to go to the talk on Electronic Design Automation (EDA). Peter Collins, manager of software services for Exemplar Logic, headed the BoF and talked about how his company had done an NT port but now had a Linux port. He pointed out that EDA grew up on Unix-based systems like Suns and the capabilities of Linux were a better fit for current EDA users.

The Trade Show

The trade show started on Wednesday. While this was not a Linux-specific trade show, Linux had a large presence. Linux vendors included Caldera, EST (makers of the BRU backup utility), InfoMagic, Linux International, Red Hat, Walnut Creek CDROM, WorkGroup Solutions and Yggdrasil. Plus, of course, our booth where we were giving away sample copies of *Linux Journal*. Lots of other vendors came by to talk about Linux and the Linux products they sell.

Linux interest was very high. While Usenix is a geek conference, these are mostly professional geeks who are making serious technical decisions for real companies. I answered many “It seems like Linux could do this” inquiries.

Within the trade show I think SSC offered the biggest hit. We just finished our new “fences” T-shirt. We sold out of the shirts in about four hours on the first day. This gave me the feeling that I was at the right show—not one where Microsoft was being honored.

Linus Talks and Linux Talks

On Wednesday afternoon we proved how significant the Linux interest/ presence was. Linus was scheduled to talk on the future of Linux in a fairly large room, which soon filled up, with standees everywhere—including the hall outside. Usenix quickly offered to move the crowd into a much larger hall.

The talk went well as Linus explained new features and new ideas. I won't bore you with details. The important thing is that the goal is world domination. To some this sounded like humor. Maybe it was. Only time will tell. In the meantime, building a superior product can't hurt.

Wednesday evening was a time for more Linux sessions. I attended one called *The Classroom of the Future* that showed how an experimental program brought the Internet to K-12 schools in Ireland. I also attended another called *The Future of the Linux Desktop*, missing Greg Wettstein's talk on perceptions. [see Greg's article “Linux in the Trenches” in *LJ* #5, September 1994—Ed.]

Thursday was another day of talks and trade show. Peter Struijk, SSC's “head nerd” managed to make it to Victor Yodaiken's presentation on real-time Linux [see *LJ* #34, February 1997] and a talk on the /proc file system by Stephen Tweedie. In the evening, I hosted a session on embedded, turnkey and real-time systems and intended to make it to Developing Linux-based electronic markets for Internet Trading Experiments but ended up talking with some of the attendees of my session instead.

The evening ended with a short talk about Linux and reality with Stephen Tweedie and then a trip back to the hotel room to finish up this column. Then, if I run out of things to do I may actually get some sleep.

Friday offered a day of Uselinux business talks. However, the combination of editorial deadlines and exhaustion means you won't get to read about it here.

What Next?

It was a great show. Usenix has always been a great show offering high-quality sessions and a really nice mix of “non-suites”. Having Usenix/Uselinux made it all the better. I am sure there will be serious cooperation between Usenix and Linux International to continue to making Linux a big part of Usenix.

If I have one complaint it was that there was too much to do. Add a Linux International board meeting to a schedule that included sessions, talks and BoFs from 9AM to 11PM with parallel Linux tracks plus the normal Usenix tracks and there just wasn't time to breathe or, more importantly, sit down with a beer and talk to fellow kernel hackers, systems administrators or vendors.

Anyone who wants copies of the Proceedings of this conference, or to find out what the future holds with regard to Usenix, should contact USENIX Association at office@usenix.org, check out their web site at http://www.usenix.org or, if all else fails, call 510-528-8649. Oh, and if you don't know what 8649 spells, you must be new to the Unix community.

LINUX JOURNAL

Using Linux at Lectra-Systèmes

Pierre Ficheux

Issue #36, April 1997

Linux as used in CAD/CAM design by Lectra Systemes, France.

Lectra Systèmes is one of two world leaders in the design and creation of the CAM solution, CAD/CAM and cutting machines, mainly for the footwear and apparel industry. The headquarters of this company are in Cestas, in the suburbs of Bordeaux, France. Five hundred people work here, 150 of whom are in the Research and Development department.

I am in charge of systems development in the R&D Department. The system group does all developments that concern base systems (e.g.,installation procedures, graphic libraries, tools).

Since the 1980s, Lectra has developed its own computers based on Motorola 680x0 processors. The main part of the installed systems (approximately 3000 customers, 80% abroad) uses a mono-task, proprietary operating system, written in 680x0 called MILOS for “Micro Lectra Operating System”.

Why Use Unix?

A few years ago, Lectra started to become interested in database systems requiring the use of a more powerful system that would be multi-task and multi-user. After some teething problems with the Unix-like, the choice turned to implementing Unix System V3.2 for 680x0 architecture. The small team of which I am a member has managed to port the UniSoft sources as well as the X Window System graphic environment.

Lectra then decided to develop a new line of computers based on 68040 processors, much more powerful than the 68030. The operating system used was the Unix USL SVR4.0 version, and another port was made.

Although this task proved to be very interesting, we were persuaded that this computer (named OpenCad) would be the last one designed from scratch by the R&D teams. A few people continued to show interest, but continuing to support a series of computers that were too small to be competitive made it difficult to remain in a hardware market that is a race against power and low prices.

Lectra's Cultural Revolution

Despite OpenCad's commercial success with our customers, Lectra's management quite rightly decided to launch the development of a completely new range of products utilizing mainly Intel 486 and Pentium architecture, still with a Unix environment and X Window System. The database applications which use many resources would, on the other hand, be targeted to SUN SPARC architecture.

After some comparative tests between the different versions of Unix on the PC, it was decided to use Linux, which proved to be sturdy, have high performance, and the right price. Also, having the sources of the system available proved to be advantageous, as we use many special peripherals for which the adaptation would be much more difficult on a Unix machine.

Developments under Linux

Having chosen the system, we now needed to adapt Linux to an industrial solution. It is quite clear that Unix (and, therefore, Linux) is slightly more difficult for a final operator to use. This adaptation must be done in two stages:

- at the installation procedure of the final product, as it is not possible to expect a technician (a customer) to know how to install Slackware
- at the user interface, so that the administration of the station base (network, users, access rights) and the specific functionalities of Lectra are easily accessible by someone who is not necessarily a computer scientist

Installation Procedure

The Lectra distribution uses the same principles as other distributions—two boot floppies and a CD-ROM. The installation screens use dialog-0.3, which has proved to be extremely simple and powerful when it comes to creating a series of installation screens. The main Lectra Linux installation window can be seen in Figure 1.

Figure 1. Lectra Linux Installation Main Menu

The main advantage when choosing Linux in this domain is that it has the possibility of creating an extremely precise installation procedure (i.e., only what is required is installed), and it is therefore very quick. The current Lectra Desktop version takes less than 10 minutes to install on a Pentium 120. In comparison, the same desktop version on a Solaris system takes nearly an hour, as it is necessary to install the Solaris CD first, followed by Solaris patches, and *then* the Lectra Desktop.

The different packages are managed as ISO-9660 files (with Rock Ridge extensions) from a Linux structure using the mkisofs program. The ISO images are then written on the master CD using a PC under Microsoft Windows.

OpenPartner Desktop

The first graphic applications under MILOS had a very spartan look, due to the weak performances of the graphic controllers at that time (beginning of the 1980s). The screens, although graphical, could manage only 16 colours, and moreover, they did not use multiple windows. Hastened by competition, in 1990 Lectra decided to develop the graphic interfaces to a multi-window system facilitating the operator's work for basic operations, such as launching applications or working with files. This tool, called OpenPartner, was initially developed for the MILOS target using a low level owner library (similar to Xlib calls). The structure of the interface seemed very similar to that of the Xt/ Intrinsics Widgets hierarchy.

The port of OpenPartner to the Unix environment comes with the addition of the administrative functions of the station by a privileged user, in particular:

- Adding and removing Lectra packages
- The network management (adding/removing stations, NFS mount management)
- Serial lines and modem management
- Printer management
- Licenses and Lectra applications management
- User management, in particular the applications authorised for each one

Figure 2 is an example of the main window in the OpenPartner environment with the package management utility, P-Manager.

Figure 2. OpenPartner with the Package Management Utility P-Manager

One of the important tasks was to develop a printer management system that could be extended and was easy to use. Even while supporting Unix, we have to admit the print system on MS Windows or even MacOS is much clearer and

easier to use than on our favourite operating system. Furthermore, all the printers currently on the market are supplied with their own Windows or MacOS driver.

Our print system (operating customer/server) uses the Ghostscript program which manages different types of printers (PCL, PostScript, raster) on varied connections—serial line, Centronics, network, SCSI. A graphic tool integrated in OpenPartner, I-Manager, is used to select only the printer characteristics that are actually used. The list then appears in the print selector of Lectra applications.

Figure 3. Integrated Graphic Utility I-Manager

Lectra applications

The applications are intended for industrial professionals in apparel. The garment is designed in various stages and corresponds to different trades. One of the characteristics of the apparel industry is the use of sub-contractors and delocalisation. Various stages of the apparel might be realised by the apparel maker, yet production could be sub-contracted to another country. Some countries deal only with the design or the production, and supply several major brands.

The result of this situation for a company like Lectra is that it is absolutely essential to design open software, as very few customers will buy a complete series, and it is therefore necessary to know how to communicate with competing software.

Another important constraint is to support lots of languages, such as Japanese, Chinese or Russian, by using tools such as the Asiatic front-end-processor under X11.

On first approach, we can expect the following stages when designing a garment.

Design

The designer has to create a garment model, like an artistic drawing. His/Her work is mainly based on the choice of shapes, colours, and types of fabrics that can be used. The advantage of having a data processing tool is clear. Other than the possibility of working on an “electronic sheet”, the software enables the pattern maker to import fabric motifs in an electronic form or even by using a scanner, to file the suggestions of different collections, and to make fabric simulations in 2D or 3D.

The ProStyle software offers all the above functions on a high performance Linux PC (Pentium 120 with Diamond Stealth S3-968 graphics board, 16 million colours). Sublimation printouts are also available. The software also works with the Silicon Graphics architecture.

Figure 4. Graphics Software ProStyle

Pattern Making

The pattern makers must create the pattern of the garment, i.e., the plan with quotation, from the information given by the designer. He/She must also manage the different sizes, or grading, available. The information in this phase is one of the most interesting with the marker (see below) as it has a high turnover—the number of patterns produced.

Figure 5 is a view of the initial screen of the Modaris application, designed for the pattern maker.

Figure 5. The Modaris Pattern Maker Utility

The Marker

The marker maker must optimise the material use, i.e., the piece of fabric called width, depending on the list of pieces given by the pattern maker. The quality of the work of a marker maker is expressed in the efficiency of a marker, which corresponds to the material quantity used in relation to the material loss. A good marker has an average efficiency of 85%, meaning 15% of the material is lost. A gain of a few tenths of a percent in production can have important economic consequences when it concerns costly materials like leather and high quality fabrics.

The Diamino software working on Linux/PC has a semiautomatic marker, making work easier for the marker maker as it facilitates positioning the piece. It also has a new automatic marking module (which marks all the pieces on the width without any manual intervention), the performance of which today is nearing 2% of that of a professional marker.

With the current PC architecture, it is now possible to obtain such a result at a very attractive price, whereas only a few years ago, the existence of an efficient module for the automatic marker was impossible due to material costs.

Figure 6 shows the evolution of the automatic marker making performances depending on the architecture used (time in seconds). The old Lectra X400 and X410 computers are based on the 68030 and 68040 processors operating

under System V R4. The three PCs—DX2/66, Pentium75, Pentium120—used Linux kernel version 1.2.13.

Figure 6. Relative Performance of Automatic Marker Software on Various Platforms (time in seconds)

The Cut and Plot

The cut of the pieces can be done by hand or with a cutting machine, which has a better turnover advantage. In some cases, one does not cut the pieces, but just plots the shapes on a paper support to give to the sub-contractor.

The previous range of products from Lectra required the purchase of a computer per plotter or cutter, as the MILOS operating system of the computers was mono-task; in other words, the plotters and cutters were run by serial channel.

The new VigiPrint software developed under Linux controls about ten plotters simultaneously, giving a cost savings that should not be ignored when considering the configuration, together with much more facility in controlling the plot by managing the plotters on the same screen. The number of cutters managed by the software is limited to only one for safety reasons—the operator must be attentive to any blade break or other anomaly. The cut is made either with a blade (most common), a high pressure water jet (2000 bars), or a laser beam.

The Production Management

The Lectra suite of software, MasterLink and StyleBinder, enables managing all the data manipulated by the previous trades. In this way, it is possible for a given product to define the production follow-up folders which are filed in the relational databases. These databases make it possible to file various elements of a previous collection and re-use them in a current collection.

Below is a screen from the StyleBinder software under Linux.

Figure 7. StyleBuilder Software under Linux

Problems Encountered

The main problem is the integration of new peripherals, since the PC world is literally in the hands of Microsoft. Some peripheral manufacturers, mainly small manufacturers, are attentive to the Linux evolution and collaborate easily when designing drivers. The large companies are much more difficult to convince, and they often hide behind the imperatives of “marketing strategy”, refusing to

supply the information required. Generally they refuse to accept any solution other than Microsoft, especially when it concerns technical support, since the development teams are rarely directly accessible in such structures.

However, the problems encountered are limited in number:

- The MATROX Millennium board which was not supported by XFree86 because MATROX requires a non-disclosure agreement before they will release the technical information required. This situation is a hindrance for us, since the Lectra applications use specific peripherals, e.g., graphic tablets, miniature keyboards, ultrasound pens, which mean the X server must be altered to manage the “input extensions”. Our choice is therefore limited to boards based on S3 circuitry, like Diamond and #9.
- Lots of storage peripherals connected via a parallel port can't be used with Linux, as each port is unique to each manufacturer and the information and protocol used are extremely difficult to obtain. Therefore, we use SCSI peripherals and internal drives on floppy port (ftape).
- The Linux installation on the PC Notebook is sometimes difficult due to the special graphic circuits not supported by XFree86.

Other subjects have caused us, and in some cases are sometimes still causing us, a few worries:

- The absence (at the moment) of a Microsoft Windows emulator at a *professional* level is a serious problem since our customers sometimes need to use documents from MS Windows to integrate them into our applications. The next software release of this type will be a boost for the generalisation of Linux as a desktop solution.
- The swap management of Linux 1.2.13 does not seem quite as good as some other versions of Unix (such as SunOS-4.1.3). It appears the swap operating on Linux 2.0 has improved greatly.
- The Lectra environment uses the SCSI interface intensively (for some top-of-the-line printers and scanners), and we have corrected a few bugs in the Linux SCSI driver. The corrections forwarded to the Linux developers seem to be integrated in the 2.0 kernel.

Conclusion

The experience of the new Lectra range shows it is possible to build an industrial solution under Linux. The system is stable and powerful. It is possible to gather a wide range of information thanks to the Internet, which has proved to have the highest performance in technical support. Having system source code makes it possible to develop more functions more easily (e.g., material drivers or individual protocols, specific file systems).

The new orientations announced during the last Linux congress in Berlin (improvement of the Virtual File System, optimisation of the EXT2 file system, the Wabi MS Windows emulator, multi-processor support, adoption of Linux by Digital) assures us Linux has acquired much industrial maturity.

Furthermore, the choice of a PC platform allows us to offer customers industrial and administrative applications on the same machine.

The opinions expressed in this article are those of the author and not of the Lectra-Systemes company.

Pierre Ficheux is in charge of the system development team at the R&D department of Lectra-Systemes, Cestas, France. He has been a Linux enthusiast (some say fanatic) since version 0.99. When not doing something with Linux, X, HTML or fr.comp.os.linux, he loves picking tunes on his guitar or just staying on the nice beach of Arcachon. He can be reached by e-mail at: pierre@rd.lectra.fr.

Archive Index Issue Table of Contents

Advanced search

A 10-Minute Guide for Using PPP to Connect Linux to the Internet

Terry Dawson

Issue #36, April 1997

Having trouble connecting to the Internet? Here's an easy way to do it using PPP.

Connecting your Linux machine to the Internet with PPP is easy in most situations. In this article I show you how to configure PPP for the most common type of connection. We assume your Linux machine is a stand-alone machine that dials into an Internet Service Provider and performs an automatic login, and the Internet Service Provider allocates the IP address that your machine will use. You can find details of how to configure PPP for other situations in the PPP-HOWTO by Robert Hart. You will need the right software and a couple of pieces of information before you start. Let's get started.

Preparation

First, check that you have the right software. The program that manages PPP for Linux is called **pppd**. The pppd program is linked very tightly with the kernel, so you must run a version of pppd that matches your kernel.

```
Kernel Version    pppd version
1.2.*             2.1.2d
1.3.0 -> 1.3.84   2.1.2d
1.3.84 -> 1.3.99  2.2.0f
2.0.*             2.2.0f
2.1.*             2.2.0f
```

Check the version of pppd and kernel that you have installed with the following commands:

```
$ /usr/sbin/pppd version
$ uname -a
```

The first command is a trick. The pppd command doesn't actually have a version option. However, the version number will appear in the error message pppd returns, since you have supplied it with a bad argument.

If the first command fails, you probably don't have PPP installed. You can obtain the latest version of the source from:

```
ftp://sunsite.unc.edu/pub/Linux/system/Network/serial/ppp/
```

If you have installed from a distribution such as Debian, Red Hat or Slackware, the pppd program is available precompiled within those distributions. You just have to get the package and install it.

Next you must check that your kernel has PPP support. Do this by giving the command:

```
$ dmesg | grep -i ppp
```

You should see the following messages:

```
PPP: version 2.2.0 (dynamic channel allocation)
PPP Dynamic channel allocation code copyright 1995 Caldera, Inc.
PPP line discipline registered.
```

If not, PPP may have been installed as a module. Become root and try:

```
# insmod ppp
```

If that fails, you will have to rebuild your kernel with PPP support. Follow the instructions in /usr/src/linux/README, and when configuring your kernel ensure that you answer “Yes” to:

```
General setup  --->
        [*] Networking support
Network device support  --->
        [*] Network device support
        <*> PPP (point-to-point) support
```

These prompts may be different in non-2.0 kernels.

Next you must note what keystrokes you will send and what prompts you will receive to log in to your ISP. The best way to collect these is to try manually logging into your ISP using a terminal program such as minicom. Be sure to make note of the capitalization of prompts such as the “login:” prompt as this will be important later.

A typical scenario follows:

```
Expect      Send                Comment
------      ----                -------
nothing     AT&F/r              (mode reset)
OK          AT&D2&C1/r          (mode initialization)
OK          AT&D555-9999/r      (modem dialing command)
```

The modem dials, sends CONNECT message and then you enter userid and password as follows:

```
login:  username/r
password: password/r
```

Lastly, you must know the IP address of a nameserver so that you can configure your name resolver and use host names instead of IP addresses. Get this information from your ISP.

Configuring PPP

The pppd program can accept configuration parameters from two places. The first is from the command line, and the second is from “options” files. The arguments supplied are close to identical in either case, but the command line method can be messy. So I will describe how to configure PPP using the options files instead.

The normal location of the options file is:

```
/etc/ppp/options
```

The options file is a simple text file containing parameters pppd will use when it is executed—one parameter per line. The options file must be readable by whoever will execute the pppd program. In most installations this will be root, either directly or by executing pppd from a program like sudo.

If you don't have an **/etc/ppp** directory, as root create one using the following commands:

```
# mkdir /etc/ppp
# chown root:root /etc/ppp
# chmod 755 /etc/ppp
```

Create an **/etc/ppp/options** file that looks like the following example:

```
debug
/dev/ttyS0
38400
modem
crtscts
lock
connect /etc/ppp/net-connect
asyncmap 0
defaultroute
:
```

This example assumes:

1. You want PPP to give you diagnostic information as it runs.
2. Your modem is connected to serial device /dev/ttyS0.

3. You want the serial port speed to be set at 38400 bps.
4. You want to listen to the Data Carrier Detect signal.
5. You will use hardware (RTS/CTS) handshaking.
6. Your dialer program is /etc/ppp/net-connect.
7. You have a full 8 bit clean connection.
8. By default datagrams should be sent via the PPP link.
9. You want the PPP server that you call to assign the IP address you will use.

These are all fairly typical defaults for an ISP connection. You will have to adjust the serial device to suit where you have your modem connected and, if you are using data compression, you might want to set your serial port speed to something higher. PPP provides a means of *escaping* select characters, so that they do not interfere with your connection. For example, if you were running PPP over a link that would disconnect if it received a control-D character, you could ask PPP to escape that character, and it would automatically replace it with another and reverse the process at the other end. While the default is safe, it escapes a number of characters that normally don't need escaping and this will decrease the performance of your link. Since most ISPs provide 8 bit clean links you don't need to escape any characters, so we tell pppd not to, using the asyncmap option.

The pppd package includes a program called **chat**. The chat program is a simple program that can be used to automate the dialing procedure. The chat program also accepts arguments from the command line or from a file. Again I'll describe how to configure it from a file as this is the better method.

To make use of the chat program from within pppd, we must ensure that the **connect** option points to a script that calls chat. Create a script called **/etc/ppp/net-connect** that looks like:

```
#!/bin/sh
/usr/sbin/chat -v -t 60 -f /etc/ppp/net-chat
```

This shell script will invoke the chat command with the **-v**, **-t** and **-f** arguments. The **-v** argument is useful when you are configuring pppd, as it sends verbose diagnostic messages to the system log to show you what is happening as the chat program runs. The **-t 60** argument simply tells the chat program to wait 60 seconds for the expected text to arrive before timing out with an error. The **-f** argument tells chat the name of the file it should use to get the expect/send sequences it will use to login.

Make sure the script is readable and executable by whoever will invoke pppd. Assuming again that “whoever” is **root**, use the following commands:

```
# chmod 500 /etc/ppp/net-connect
# chown root:root /etc/ppp/net-connect
```

Create a chat script called **/etc/ppp/net-chat** that will automate the login sequence as described earlier. I will base this script on the details presented in the table.

```
ABORT "BUSY"
ABORT "NO CARRIER"
""              AT&F\r
OK                AT&D2&C1\r
OK                ATD555-9999\r
ogin:
sword:
```

The first two lines are special. The **ABORT** keyword is a special token that allows you to specify strings of characters that will cause the chat program to exit. In the example presented, if the chat program receives either the string **"BUSY"** or the string **"NO CARRIER"** then it will abort immediately. The rest of the file is a simple list of expect/send pairs, based on the information we gathered when we manually logged in. The above example reads in full:

> ABORT the script if we receive **"BUSY"** or **"NO CARRIER"**. Expect nothing, then send **AT&F< carriage-return>** to reset the modem to factory configuration, expect to receive **OK** then send **AT&D2&C1<carriage-return>**, then expect **OK** and send **ATD555-9999<carriage-return>**, then expect **login:** and send **username<carriage-return>**, then expect **sword:** and send **password<carriage-return>**, and then exit normally.

There are a couple of important points to note in this example. First, the modem initialization string I've suggested will, in most modems, ensure that the modem will raise the Data Carrier Detect line when a call is connected, and will hang up the call if the DTR line is lowered. This ensures that the modem is matched with the **modem** option supplied to pppd. Second, I haven't used the full prompt, but only the last few characters. This is generally good practice because under some circumstances the first characters from a line may be dropped. Looking only for the last few characters ensures our login succeeds even if this occurs. Finally, you will notice the <carriage-return> is coded as **\r**. There are a range of other characters may be encoded and sent in this way, if necessary. The chat man page explains what they are should you need to use them.

Finally, we must ensure this script is readable by whoever will invoke pppd. Again assuming that whoever is be **root**, you can use the following commands:

```
# chown root:root /etc/ppp/net-chat
# chmod 600 /etc/ppp/net-chat
```

Configuring the Name Resolver

The name resolver is a small piece of software within the standard Linux library that allows automatic conversion of a host name, e.g., sunsite.unc.edu, into an IP address, e.g., 152.2.254.81.

Configuration of the name resolver is easy; there is only one file to change. You will almost certainly already have this file on your machine, but you will need to configure the correct address for the nameserver. Assuming your ISP supplied you with a nameserver address of **128.78.64.10** then your **/etc/resolv.conf** file should have a line that says:

```
nameserver 128.78.64.10
```

Starting the Link

To start the PPP link, all you need to do is execute the following command as root:

```
# /usr/sbin/pppd
```

The pppd program will start and will search for its options in the standard locations. It will find our options file at /etc/ppp/options and read each line. When it has finished processing all available options, it will open the specified serial device, create a lock file to prevent other programs from trying to use it, and then attempt to run the connect program and to execute the /etc/ppp/net-connect script. The net-connect script will execute the chat program telling it that it should take its parameters from the /etc/ppp/net-chat file. The chat program starts, reads each of the lines from the net-chat file, waits for the strings, and sends the responses it has been given. Provided the chat program did not **ABORT** then control is passed back to the pppd program, which will then switch the line into PPP mode and create a PPP network device. The pppd program will automatically begin negotiation of some configuration details with the PPP program at the other end of the link. The most important of these details is the IP address you will use. The pppd program will create a ppp network device **ppp0** and then configure it with the details it obtained from the other program. Finally, the pppd program will configure your routing table with a route that tells your Linux machine it should send datagrams to the PPP link, if it doesn't have anywhere better to send them. The pppd program will then sit happily in the background until either the line fails, the remote end closes the connection or you terminate it locally.

Okay, that sounds complicated, so a summary:

1. pppd starts.
2. pppd reads /etc/ppp/options.

3. pppd executes /etc/ppp/net-connect.
4. chat reads data from /etc/ppp/net-chat.
5. pppd obtains IP address details from server.
6. pppd creates ppp0 device and configures it.
7. pppd creates default route.
8. pppd runs in background.

Testing the Connection

To test the connection, do each of the following steps in turn.

Step 1:

```
run /sbin/ifconfig
```

The ifconfig program is used to set or display network interface configurations. Here you are interested in displaying only.

Step 2:

```
# /sbin/ifconfig
```

The output should look like Listing 1.

Listing 1

The **inet addr** field is the IP address you have been allocated. The **P-t-P** field is the address of the PPP machine at the other end of the link. This means your PPP network connection has been successfully established.

If you don't see a **ppp0** device, check your system log file, i.e., /var/adm/ messages, to ensure that your chat script worked successfully. Correct any possible errors. If you see any nasty looking error messages, double check that you are using the correct version of PPP for your kernel.

Step 3: ping the PPP Remote Host. The ping command sends specially formatted datagrams to a host that that host will send replies to. This allows us to check that we have a working route to that host. Listing 2 shows our case. Those "**64 bytes from ...**" lines in the listing mean we are talking successfully to the machine at the other end of the link. This is good, since it means the link is working.

Listing 2

If you don't see any of the "**64 bytes from ...**" lines, this means you are not properly talking to the remote machine. Double check your chat script and the system log file.

Step 4: ping your nameserver. This is an important test to be sure the default route pppd put in place is working. To do this, ping the nameserver address configured into the /etc/resolv.conf file. In our case:

```
# ping 128.78.64.10
```

Output will be similar to what you observed when you pinged the PPP server.

Listing 3

If this test fails, it could mean your default route hasn't been added properly. To double check, run the route command as shown in Listing 3. The route command displays the contents of the IP routing table. The **-n** option tells it not to try and convert IP addresses into host names. The line starting with **0.0.0.0** is the default route. If you don't see a line like this, double check that you have included the **defaultroute** option in the /etc/ppp/options file. If you have a line like this but it doesn't point to ppp0, check that your system isn't already creating a default route to another device. If it is, find which **rc** file is doing it and comment out this entry.

Step 5: ping a remote host. This is the real and simple test. Try either:

```
# ping sunsite.unc.edu
```

or:

```
# ftp ftp.funet.fi
```

If this works, you are connected properly to the Internet. Enjoy.

If the command just sits there and, after a minute or so, gives you an error message about being unable to resolve the host name, check that you have modified your /etc/resolv.conf file correctly, and that the IP address you have configured there is the correct IP address for your ISP's nameserver.

Dropping a Connection

To drop a connection you just need to kill pppd. When it exits, it will hang up the line, if you've configured the modem as I've suggested.

On most distributions this will be as simple as:

```
# killall -HUP pppd
```

Making PPP Automatically Redial

If you are lucky enough to have a semi-permanent connection to your ISP, i.e., one where you can stay connected for as long as you like, you may want to have your Linux automatically redial if the telephone call drops out for some reason. Here is a simple way of doing this that assumes you have configured your PPP link to be activated by root.

The first very important step is to add this line to your /etc/ppp/options file:

```
-detach
```

This line tells pppd **not** to go into the background after it has successfully connected. The next step is to add a line to your /etc/inittab file that looks like this:

```
pd:23:respawn:/usr/sbin/pppd
```

Put this line down with the other lines that are similiar to it—the ones that run the login program.

This line simply tells the init program that it should automatically start the /usr/sbin/pppd program and that it should automatically restart it if it dies. Provided you have your modem configured to raise Data Carrier Detect and you have configured pppd as I have described, **init** will ensure the pppd program is always running and re-run it if it terminates.

A word of warning—this is simple, but provides no safeguards against problems that might cause the telephone call to be successfully made and then hang up. If you experience this problem, the init program will quite happily keep re-running the pppd program until you tell it to stop. You could run up quite a telephone bill if something nasty goes wrong.

Conclusion

This article describes a basic PPP configuration. There are many excellent documents that provide more detailed and comprehensive information on the subject. This article should be sufficient to get you connected to the Internet in a typical configuration. If you have any problems you cannot diagnose, I strongly recommend you read the PPP-HOWTO by Robert Hart at:

```
http://sunsite.unc.edu/LDP/HOWTO/PPP-HOWTO.html
```

Robert has done an excellent job in rewriting the HOWTO, and it should be of assistance to you.

Terry Dawson is the author of a number of Linux HOWTO documents including the AX25-HOWTO, IPX-HOWTO and the NET-2-HOWTO. Terry has been an advocate of Linux from the moment he booted Linux 0.12 and saw the potential for Linux to significantly enhance experimentation in networking protocols in Amateur Radio. He can be reached via e-mail at terry@perf.no.itg.telecom.com.au.

Archive Index Issue Table of Contents

Advanced search

od—The Oddest Text Utility Around

Randy Zack

Issue #36, April 1997

When you need to debug binary code, use a dump provided by a nifty command called od.

Suppose you are writing the next great spreadsheet for Linux, and you're actually getting along pretty well. You have a program that can edit cells, format the screen, and do all the really good spreadsheet stuff. And you can even save the sheets in a user-specified file. But then you make a change to the format of the file, and you realize you need to examine the file, byte by byte, in order to determine what went wrong with that last change. You know that Emacs can show you the file, but you can't remember exactly how to get into hexadecimal mode, or what to do once you are there.

Or suppose you are writing a viewer program for your favorite word-processor, which runs only under your second favorite operating system (WINE and DOSEMU notwithstanding). So you need to figure out exactly what each binary code in that .wpd file really is, so that you can determine what each binary code does by trial and error. (What a trial and error process **that** would be.)

Or maybe you are curious about exactly what escape sequence is sent to your terminal when a curses program positions the cursor. (Maybe this example is a bit contrived, but it's interesting nonetheless.)

If any of these scenarios describes your current dilemma, then od is just the utility for you. od stands for Octal Dump, because it was named before computer users started using hexadecimal for everything, and because it can dump a file (binary or not) into almost any form you can imagine.

So let's see what can be done with od. The easiest thing to try is to get an octal dump of od itself. Listing 1 shows the first 6 lines of output of the command **od `which od`**. There are several things to notice about this example. (Note: I'm using an older a.out version of od, so this is might not be exactly what you see

on your system.) By way of explanation, the first column is the “offset” into the file, and the remaining columns are the actual data in the file.

There are three things to notice about this listing. First, all the numbers are in octal, or base 8. I'm not aware of anyone using octal notation for anything anymore. And of course, with the GNU version of od, there are options for changing how everything gets displayed—more on that later.

Second, all the numbers are 16 bits wide. Since Linux is a 32-bit operating system, this is probably not what you want. Again, there are ways of modifying this behavior.

Third, the third line of output contains a single *. This is od's way of saying that there are many lines just like the previous line, which have been removed from the output. It then continues the output at offset (octal) 2000, which is the first line that differs from the previous line. (Can you guess that this behavior can also be modified? It can.)

As mentioned earlier, od has many options for formatting the output. The first one to mention is **-t xS** or **-t xL**, which will cause the output to be in hexadecimal (base 16). The **S** or **L** modifier tells od to read 16 bits (**S**) or 32 bits (**L**) at a time. To all you C programmers, yes, those modifiers stand for “short” and “long.” There are other modifiers as well, and good descriptions for them can be found in the man page for od. Listing 2 shows the first six lines of output of the command:

```
od -t xS `which od`
```

od can also output the characters of the file. And if you want to do some comparisons, you can intersperse the hexadecimal output with the character output. Just give both types on the command line (see Listing 3) as:

```
od -t xS -t c `which od`
```

There are a couple of things to note about this example. The character-type arguments don't take a size modifier—they just read one character at a time. That's why we used **-t c** and not **-t cS**.

Also, the ordering of the character data looks strange. The first 4 bytes in the hexadecimal dump are **010b 0064**, while the first 4 bytes in the character dump are **\v 001 d \0**. This is because my Linux machine runs on an Intel-based chip set, which is a little endian architecture. Other architectures will print this differently. In fact, this is the easiest way I know to determine whether the machine you are running on is big-endian or little-endian. The actual command to determine this would be something like:

```
echo abcd | od -t xS
```

A little-endian machine would output:

```
0000000 6261 6463 000a
```

while a big-endian machine would output:

```
0000000 6162 6364 0a00
```

I haven't actually seen Linux on a SPARC or a DEC Alpha chip; I would guess these Linux systems would be big-endian.

Let's get back to the last example. Notice that the character output of the last example has a lot of backslashes in it. This is one method od uses to show that the character it is trying to print is really not a printable character. Another method is to show the character in octal. Examples of the first method are **\v** and **\0** and (at offset 2024). Examples of the second method are **001** and **315** (at offsets 0001 and 2017 respectively). (Offsets are still in hexadecimal—we're getting to that problem.)

If you really hate octal, and want to see the offsets in a different base, od allows that. The option is **-A x** to see the offsets in hexadecimal, or **-A d** to show the offsets in decimal. (Enough of showing listings of these commands—just do it.)

You might have noticed that od always shows 16 bytes per line. Of course, you can change this as well, by using the **-w** option. The argument after the -w flag is the number of bytes to read before outputting a line of text. The default without the -w flag is 16 (as you can see from all the examples). The default with the -w flag (i.e. **-w** by itself) is 32. Unfortunately, I couldn't get this option to work on my machine. Every number I gave (-w20, -w18, -w16) caused od to report “invalid width specification.” (I'm using GNU textutils version 1.9, for what it's worth.)

Sometimes you want to see the whole file, and not repress any output. The **-v** option tells od to not skip any lines, and to output everything. This can be useful if you need to compare two different binary files, and you want to compare the actual bytes in the files, without skipping any of the output.

Finally, all of these options have a long format, as is standard with GNU utilities. For example, the **-v** switch can be expanded to **--output-duplicates**. I tend to use the long form in scripts, so it is clear to others exactly what options I'm sending to the program, and the short forms when I'm just working.

So, how exactly do you see the escape sequence sent to your terminal when a curses program positions the cursor? Try the command:

```
tput cup 10 10 | od -t c
```

Randy Zack can be reached via e-mail at randy@acucobol.com.

Archive Index Issue Table of Contents

Advanced search

New Products

M. L. Richardson

Issue #36, April 1997

Raima Database Manager, PC Watchdog, TextSurgeon 2.0 and more.

Raima Database Manager

Raima Corporation announced that the Raima Database Manager++ database is now available for Linux. RDM contains low level C and C++ APIs and support for the relational database model, pointer-based network database model, and combined database model. Buyers of the development license can distribute run-time copies free. Contact Raima for pricing of license.

Contact: Raima Corp., 1605 NW Sammamish Rd. Suite 200, Issaquah, WA 98027, Phone: 800-327-2462, Fax: 206-557-5200, URL: http://www.raima.com/.

PC Watchdog

Berkshire Products has announced the PC Watchdog, a system watchdog/monitor board. The board is a short 8-bit ISA card that monitors a PC for hardware/software lockups to ensure maximum system availability. It provides a solution for Telecom, BBS, Voice Mail, File Servers and Industrial systems that occasionally lock up, but must be available on a continuous basis. Watchdog comes with TSRs and utility programs for control of the board. PC Watchdog is available for Linux. Contact Berkshire Products for pricing.

Contact: Berkshire Products, P.O. Box 1015, Suwanee, GA 30174, Phone: 770-271-0088, Fax: 770-932-0082, URL: http://www.berkprod.com/.

TextSurgeon 2.0

SoftwareForge Inc., announced the release of TextSurgeon 2.0, a new text editing system for Linux. TextSurgeon comes with the ability to use most of the standard Linux programming interpreters such as shell, awk, Perl and sed for

automation of editing tasks. It also includes special support for code browsing, and advanced features especially for C/C++ coding. It is available for $50 US.

Contact: SoftwareForge Inc., Chicago, Ill., E-mail: unixguy@aol.com.

Edith Pro/X11

ZFC has announced the release of Edith Pro for X11, a quality, user friendly text editor. It is highly flexible and provides many features not available in other editors. For details see the web site at http://www.nl.net/~zfc/. It is available for Linux at a price of $50 US for a single host, $250 US for a site license.

Contact: ZFC/home P.O.Box 15813, 1001 NH Amsterdam, the Netherlands. Phone 31-20-4-208-248, E-mail zfc@zfc.nl, URL: http://www.nl.net/~zfc/.

Java Numeric Library

Visual Numerics has announced the release of the Java Numeric Library (JNL), a solution for developers of platform-neutral, network-centric, computational applications. The JNL is a set of numerical extensions to the Java Language for use in many scientific and research situations. For more detail and pricing contact Visual Numerics.

Contact: Visual Numerics, Inc., 9990 Richmond Avenue, Suite 400, Houston, Texas 77042, Phone: 713-954-6761, Fax: 713-781-9264, E-mail: dsayed@boulder.vni.com, URL: http://www.vni.com/.

MagicFax

Clarity Software, Inc. has announced MagicFax, software for the WWW that allows faxes to be sent anywhere in the world for free to other MagicFax users. Faxes sent to non-MagicFax users will save substantially on transmission costs, as MagicFax routes the fax to the sender's MagicFax Web Server nearest the recipient to avoid long distance charges. MagicFax is available for Unix platforms and will be available for Linux in April. For pricing contact Clarity Software.

Contact: Clarity Software, Inc., 2700 Garcia Ave., Mountain View, CA 94043, Phone: 415-691-0320, E-mail: 104436.2576@compuserve.com, URL: http://magicfax.clarity.com/.

F-Secure Data Security Software

Data Fellows has announced F-Secure Commerce and F-Secure VPN. F-Secure Commerce is a browser plug-in which allows any browser user anywhere in the world to have a fully secure, authenticated and strongly encrypted connection

to the web server running F-Secure Commerce for Server software. F-Secure VPN (Virtual Private Network) connects multiple separate LANs together into a secure virtual network solution over the Internet. It's a software-only solution which runs on a standard Intel-based PC. For pricing contact Data Fellows.

Contact: Data Fellows, Inc., 4000 Moorpark Ave., Suite 207, San Jose, CA 95117, Phone: 408-244-9090, Fax: 408-244-9494, E-mail: f-secure-sales@datafellows.com, URL: http://www.datafellows.com/.

Stronghold 2.0

C2Net Software announced the release of Stronghold 2.0, a commercial webserver on the Unix platform. The security interfaces have been redesigned and built on the new Apache 1.2 code base. Many productivity and performance enhancements have been made, and Stronghold is now fully compliant with the new HTTP/1.1 standard. For pricing contact C2Net.

Contact: C2Net Software, Oakland, CA, Phone: 510-986-8770, E-mail: cman@c2.net, URL: http://stronghold.c2.net/.

Archive Index Issue Table of Contents

Advanced search

Indexing Texts with SMART

Hans Paijmans

Issue #36, April 1997

Here is a “smart” program for indexing and retrieval of documents.

Although my colleagues here at Tilburg University may think the time I spend fiddling with Linux on a PC could be put to better use, they are wrong. The “fiddling with Linux” I do at home; at work I do only the minimum necessary to keep Linux fed and happy. As most readers of this journal know, this involves making the occasional backup and not much else.

When I sit in front of my PC, I work (well, mostly). Linux makes it possible to do my work with a minimum of fuss, and a big part of the credit for this goes to Jacques Gelinas, the man who wrote UMSDOS: a layer between the Unix operating system and the vanilla MS-DOS file allocation table. This program makes it possible to access the DOS partition of my hard disk from either operating system. This is good news, since I am totally dependent on two programs: SMART, an indexing and retrieval system, and SPSS for Windows, which twiddles the data I obtain from SMART. SMART runs only under Unix (and not all Unices, for that matter) and SPSS4Windows, obviously, runs under MS Windows. Whatever the virtues of this operating system may be, you emphatically do not want to use it in any kind of experimental environment.

I suppose **S**tatistical **P**ackage for the **S**ocial **S**ciences (SPSS) will be familiar to most Linux users. If not—SPSS is a statistical package not only for the “social sciences”, but also for anyone who needs statistical analysis of his data.

SMART is an indexing and retrieval program for text. It not only indexes the words, it also adds weights to them. It allows the user to compare the indexed documents in the Vector Space Model and compute the distances between documents or between documents and queries. To understand why this is special, we must delve a bit into the typical problems of information retrieval, i.e., the storage of books, articles, etc., and their retrieval based on content.

Why Indexing Is Not Enough

At the end of the 60s, automatic indexing of texts became a viable option, and many people thought the problems of information retrieval were solved. Programs like STAIRS (IBM, 1972) enabled the users to file and rapidly retrieve documents using any word in the text and boolean operators (AND, OR, NOT) with those words—who could ask for more? Then in 1985, a famous article was published by two researchers in the field. Footnote 1. In this article, they reported on the performance of STAIRS in real life, and they showed that the efficiency of STAIRS and similar systems was, in fact, much lower than assumed. Even experienced users could not obtain a recall of more than 20-40% of the relevant documents in a database of 100,000 documents, and worse, they were not aware of this fact.

The problem with all retrieval systems of this type is that human language is so fuzzy. There may be as many as a dozen different terms and words pointing to one and the same object, and one word may have widely different meanings. In information retrieval, this can lead to one of two situations. One, you obtain a high precision where almost all the retrieved documents are relevant, but an unknown number of other relevant documents are not included. Or two, you get a high recall that includes a lot of irrelevant documents in the result. When the proportion of irrelevant documents is high in a retrieved set of documents, the user will most likely stop looking before he or she has found all the relevant documents. At this point, his "futility-point" has been reached. In such a case the net result is equivalent to those relevant records **not** being retrieved. Therefore, the concept of ranking, i.e., the ordering of retrieved documents based on relevance, is very important in information retrieval.

SMART

Modern (and not so modern) research has offered a number of possible solutions to this dilemma, among them the concept of **weighted** keywords. This means that every keyword-document combination has a weight attached that is (one hopes) an indication of the relevance of that particular keyword for that particular document. SMART does just that: it creates indices for a database of documents and attaches weights to them. The method may be expressed intuitively as *"the more a word occurs in fewer documents, the higher the weight."* If the word "dog" occurs twenty times in a given document but in no other documents, you may be relatively certain that this document is about dogs. Information retrieval addicts like me talk about the *tf.idf* weight.

SMART offers several calculation options (I generally prefer the atc-variation, because it adjusts for the length of the individual documents.) It calculates the *tf.idf* in three steps. The first step creates the value

new_tf

for the term-frequency (*tf*) as:

$$new\_tf = 0.5 + 0.5 * \frac{tf}{max\_tf}$$

where

max_tf

is the term with the highest frequency in the document. This adjusts for the document length and the number of terms. Then the weight

new_wt

$$new\_wt = new\_tf * log\frac{N}{F_t}$$

where N is as before the number of documents and *F* is the document frequency of term *t* (the number of documents in which term *t* occurs). Finally the cosine normalization is applied by:

$$new\_wt = \frac{new\_wt}{\sqrt{\sum_{t=1}^{T} new\_wt_t^2}}$$

where T is the number of terms in the document vector. Now we have a number between zero and one that probably correlates with the importance of the word as a keyword for that document. For a detailed discussion of these and similar techniques see Salton and McGill. Footnote 2. You will love it.

When SMART has constructed the index in one of the various ways available, it can also retrieve the documents for you. This is done according to something called the “Vector Space Model”. This model is best explained using a three-dimensional example of a vector-space; you can add another few thousand dimensions in your own imagination.

Imagine you want to index your documents according to three keywords—“cat”, “dog” and “horse”—that may or may not occur in your documents. So you draw three axes to get a normal three-dimensional coordinate system. One dimension can be used to indicate the “cat-ness” of every document, the other its “dog-ness” and the third the “horse-ness”. To make things easy we use only binary values 0 and 1, although SMART can cope with floats (e.g., the “weights” mentioned before). So if a document is about cats, it scores a 1 on the corresponding axis, otherwise it scores 0. Any document may now be drawn in that space according to the occurrence of one or more of the keywords, and so we have a relatively easy way to compute the difference between documents.

Moreover, a query consisting of one or more of the keywords can be drawn in the same space, and the documents can be ranked according to the distance to that query. Of course a typical document database has thousands of keywords and, accordingly, thousands of dimensions, but the arithmetic involved in multi-dimensional distances does not matter much to modern computers.

So SMART accepts queries, ranks the documents according to the “nearness” to that query and returns them to you in that order. This ability makes it one of the best retrieval systems ever written, even though it lacks the bells and whistles of its more expensive counterparts. Although it is not really optimized for speed, it typically runs 10-30 times faster than the fastest indexing program for MS Windows that I have tried.

The DOS Connection

I do not use SMART for bread-and-butter retrieval, but for the weights it computes and the indices it creates. At this point I usually want to do some other manipulations of the data. I offer my thanks to the developers of Unix in general and to Linux in particular for creating a whole string of ever more complicated and sophisticated shell scripts, the standard Unix tools and a few of “My Very Own” utilities that suffice to process the SMART output to a file that is ready for import into SPSS.

Now I have to quit Linux and boot MS-DOS, start MS Windows and finally enter SPSS to do the statistics and create some graphs. I am a newcomer to Unix (indeed it was the fact that Linux offered a way to use SMART that pulled me over the line two years ago). While MS Windows is not my favorite operating system, SPSS gets the job done. When the output is written to disk, I immediately escape back to Linux to write the final article, report, or whatever with LaTeX.

The Bad News

On this point I have two messages—one bad. The good news is that SMART is obtainable by anonymous ftp from Cornell University and can be used free for scientific and experimental purposes. Better yet, it compiles under Linux without much tweaking and twiddling. There is also a fairly active mailing list for people who use SMART (smart-people@cs.cornell.edu).

The bad news: the manual—what manual? SMART is not for the faint of heart; after unpacking and compilation, you'll find some extremely obscure notes and examples, and that is all. Nevertheless, if you have more than just a few megabytes of text to manage **and** the stamina to learn SMART, it certainly is the best solution for your information retrieval needs. I do wish someone would write a comprehensive manual. In the meantime, you may be helped by my

“tutorial for newbies” found at http://pi0959.kub.nl:2080/Paai/Onderw/Smart/hands.html.

> This article was published previously in Issue 13 of the Linux Gazette.

Hans “Paai” Paijmans (paai@kub.nl) is a University lecturer and researcher at Tilburg University and a regular contributor to several Dutch journals. Together with E. Maryniak, he wrote the first Dutch book on Linux—already two years ago. My, doesn't the time fly? His home page is at http://pi0959.kub.nl:2080/paai.html .

Archive Index Issue Table of Contents

Advanced search

History of the Portable Network Graphics (PNG) Format

Greg Roelofs

Issue #36, April 1997

PNG is a new, awesome graphics format designed to take the place of GIF—here's how it came about.

Prehistory

The story of PNG actually begins way back in 1977 and 1978 when two Israeli researchers, Jacob Ziv and Abraham Lempel, first published a pair of papers on a new class of lossless data compression algorithms, now collectively referred to as "LZ77" and "LZ78." Some years later, in 1983, Terry Welch of Sperry (which later merged with Burroughs to form Unisys) developed a very fast variant of LZ78 called LZW. Welch also filed for a patent on LZW, as did two IBM researchers, Victor Miller and Mark Wegman. The result was—you guessed it—the USPTO granted both patents (in December 1985 and March 1989, respectively).

Meanwhile CompuServe—specifically, Bob Berry—was busily designing a new, portable, compressed image format in 1987. Its name was GIF, for "Graphics Interchange Format," and Berry, et al., blithely settled on LZW as the compression method. Tim Oren, Vice President of Future Technology at CompuServe (now with Electric Communities), wrote: "The LZW algorithm was incorporated from an open publication, and without knowledge that Unisys was pursuing a patent. The patent was brought to our attention, much to our displeasure, after the GIF spec had been published and passed into wide use." There are claims (Reference 1) that Unisys was made aware of this as early as 1989 and chose to ignore the use in "pure software;" the documents to substantiate this claim have apparently been lost. In any case, for years Unisys limited itself to the pursuit of hardware vendors—particularly modem manufacturers implementing V.42bis in silicon.

All of that changed at the end of 1994. Whether due to ongoing financial difficulties or as part of the industry-wide bonk on the head provided by the

World Wide Web, in 1993 Unisys began aggressively pursuing commercial vendors of software-only LZW implementations. CompuServe seems to have been its primary target at first, culminating in an agreement—quietly announced on 28 December 1994, right in the middle of the Christmas holidays—to begin collecting royalties from authors of GIF-supporting software. The news hit the Internet the following week; what was then the comp.graphics newsgroup went nuts, to use a technical term. As is the way of Usenet, much ire was directed at CompuServe for making the announcement, and then at Unisys once the details became a little clearer. Mixed in with the noise was the genesis of an informal Internet working group led by Thomas Boutell (Reference 2). Its purpose was not only to design a replacement for the GIF format, but a successor to it: better, smaller, more extensible and **free**.

The Early Days (All Seven of 'Em)

The very first PNG draft—then called "PBF," for Portable Bitmap Format—was posted by Tom to comp.graphics, comp.compression and comp.infosystems.www.providers on Wednesday, 4 January 1995. It had a three-byte signature, chunk numbers rather than chunk names, maximum pixel depth of 8 bits and no specified compression method, but even at that stage it had more in common with today's PNG than with any other existing format.

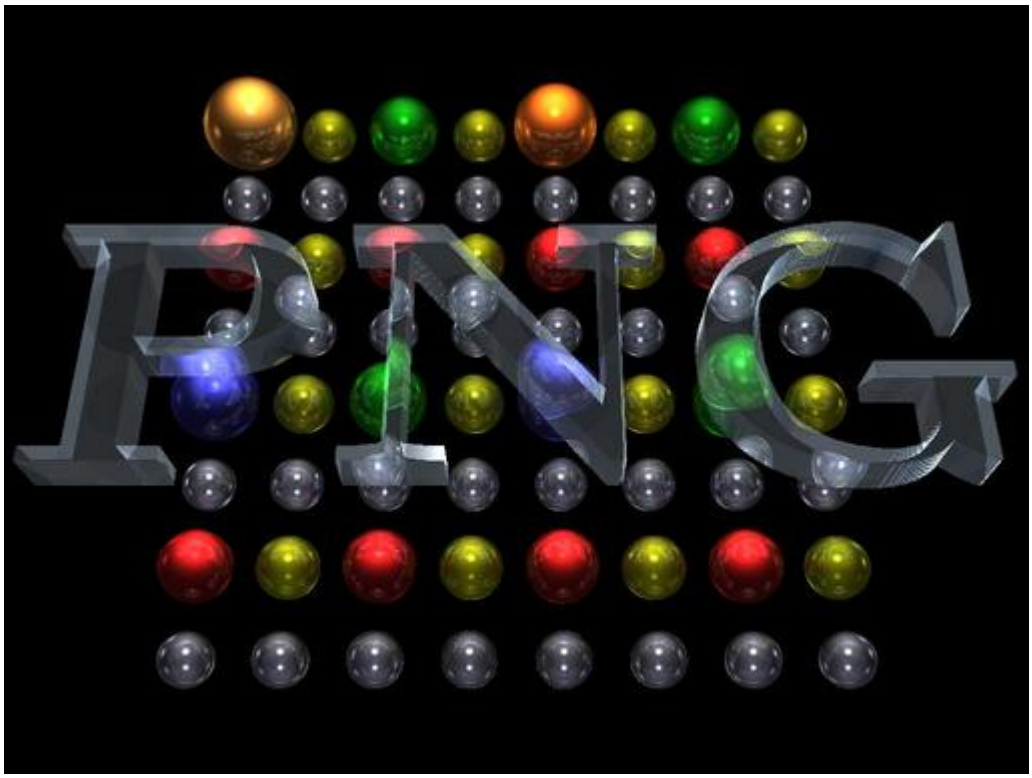
Within one week, most of the major features of PNG had been proposed, if not yet accepted: delta-filtering for improved compression (Scott Elliott); deflate compression (Tom Lane, the Info-ZIP gang and many others); 24-bit support (many folks); the PNG name itself (Oliver Fromme); internal CRCs (myself); gamma chunk (Paul Haeberli); and 48- and 64-bit support (Jonathan Shekter). The first proto-PNG mailing list was also set up that week; Tom released the second draft of the specification; and I posted some test results that showed a 10% improvement in compression, if GIF's LZW method was simply replaced with the deflate (LZ77) algorithm. Sidebar 1 is a time-line listing many of the major events in PNG's history.

Perhaps equally interesting are some of the proposed features and design suggestions that ultimately were **not** accepted: the Amiga IFF format; uncompressed bitmaps either gzip'd or stored inside zip files; thumbnail images and/or generic multi-image support; little-endian byte order; Unicode UTF-8 character set for text; YUV and other lossy, i.e., non-lossless, image-encoding schemes; and so forth. Many of these topics produced an amazing amount of discussion—in fact, the main proponent of the zip-file idea is still making noise two years later.

Onward, Frigidity

One of the real strengths of the PNG group was its ability to weigh the pros and cons of various issues in a rational manner (well, most of the time, anyway), reach some sort of consensus, and then move on to the next issue without prolonging discussion on “dead” topics indefinitely. In part, this was probably due to the fact that the group was relatively small, yet possessed of a sufficiently broad range of graphics and compression expertise that no one felt unduly “shut out” when a decision went against him. All of the PNG authors were male—a fact that is still true. (I'm sure there's a dissertation in there somewhere.) But equally important was Tom Boutell, who, as the initiating force behind the PNG project, held the role of benevolent dictator—much the way Linus Torvalds does with Linux kernel development. When consensus was impossible, Tom would make a decision, and that would settle the matter. On one or two rare occasions he might later have been persuaded to reverse the decision, but this generally happened only if new information came to light.

In any case, the development model worked. By the beginning of February 1995, seven drafts had been produced, and the PNG format was settling down. The PNG name was adopted in Draft 5. The next month was mainly spent working out the details: chunk-naming conventions, CRC size and placement, choice of filter types, palette-ordering, specific flavors of transparency and alpha-channel support, interlace method, etc. CompuServe was impressed enough by the design that on the 7th of February they announced support for PNG as the designated successor to GIF, thereby supplanting what had initially been referred to as the GIF24 development project (Reference 3). By the beginning of March, PNG Draft 9 was released and the specification was officially frozen—just over two months from its inception. Although further drafts followed, they merely added clarifications, some recommended behaviors for encoders and decoders, and a tutorial or two. Indeed, Glenn Randers-Pehrson has kept some so-called “paleo PNGs” that were created at the time of Draft 9; they are still readable by any PNG decoder today (Reference 4).



PNG Home Page Graphic

Oy, My Head Hurts

But specifying a format is one thing; implementing it is quite another. Although the original intent was to create a “lightweight” format—and, compared to TIFF or even JPEG, PNG **is** fairly lightweight—even a completely orthogonal feature set can introduce substantial complications. For example, consider progressive display of an image in a web browser. First comes straight decoding of the compressed data—no problems. Then any line filtering must be inverted to get the actual image data. Oops, it's an interlaced image: pixels are appearing here and there within each 8x8 block, so they must be rendered appropriately and possibly buffered. The image also has transparency and is being overlaid on a background image, adding a bit more complexity. At this point we're not much worse off than we would be with an interlaced, transparent GIF; the line filters and 2D interlacing scheme are pretty straightforward extensions to what programmers have already dealt with. Even adding gamma correction to the foreground image isn't too much trouble.

However, it's not just simple transparency; we have an alpha channel. And we don't want a sparse display—we like the replicating progressive method that Netscape Navigator uses. Now things are tricky: each replicated pixel-block has some percentage of the fat foreground pixel mixed in with complementary amounts of the background pixels in the block. And just because the current fat pixel is 65% transparent (or, even worse, completely opaque) doesn't mean later ones in the same block will be, too; thus, we have to remember all of the original background pixel-values until their final foreground pixels are

composited and overlaid. Toss in the ability to render all of this nicely on an 8-bit, color-mapped display, and most programmers' heads will explode.

Make It So

Of course, some of these things are application (presentation or front-end) issues, not general PNG-decoding (back-end) issues. Nevertheless, a good PNG library should allow for the possibility of such applications—which is another way of saying that it should be general enough not to place undue restrictions on any programmer who wants to implement such things.

Once Draft 9 was released, many people set about writing PNG encoders and/or decoders. The true glory is really reserved for three people, however: Info-ZIP's Jean-loup Gailly and Mark Adler (both also of gzip fame), who originally wrote Zip's deflate() and UnZip's inflate() routines and then, for PNG, rewrote them as a portable library called **zlib** (Reference 5), and Guy Eric Schalnat of Group 42, who almost single-handedly wrote the **libpng** reference implementation (originally “pnglib”) from scratch ([Reference 6). The first truly usable versions of the libraries were released two months after Draft 9, on the first of May, 1995. Although both libraries were missing some features required for full implementation, they were sufficiently complete to be used in various freeware applications. Draft 10 of the specification was released at the same time, with clarifications resulting from these first implementations.

Fast-Forward to the Present

The pace of subsequent developments slowed at that point. This was partly due to the fact that, after four months of intense development and dozens of e-mail messages every day, everyone was burned out; partly because Guy controlled libpng's development and became busy with other things at work; and partly because of the perception that PNG was basically “done.” The latter point was emphasized by a CompuServe press release to that effect in mid-June. A press release, I might add, in which their PR guys claimed much of the credit for PNG's development (sigh).

Nevertheless, progress continued. In June of 1995 I set up the PNG home page, now grown to roughly a dozen pages (Reference 7), and Kevin Mitchell officially registered the “PNGf” Macintosh file ID with Apple Computer. In August, Alexander Lehmann and Willem van Schaik released a fine pair of additions to the NetPBM image manipulation suite, particularly handy under Linux: pnmtopng and pngtopnm version 2.0. And in December at the Fourth International World Wide Web Conference, the World Wide Web Consortium (W3C) released the PNG Specification version 0.92 as an official standards-track Working Draft.

1996 saw the February release of version 0.95 as an Internet Draft by the Internet Engineering Task Force (IETF), followed in July by the Internet Engineering Steering Group's (IESG) approval of version 1.0 as an official Informational RFC. However, the IETF secretary still hasn't issued the actual RFC number at the time of this writing, six months later (sigh). The Virtual Reality Modeling Language (VRML) Architecture Group in early August adopted PNG as one of the two required image formats for minimal VRML 2.0 conformance (Reference 8). Meanwhile the W3C promoted the spec to Proposed Recommendation status in July and then to full Recommendation status on the first of October (Reference 9). Finally, in mid-October the Internet Assigned Numbers Authority (IANA) formally approved “image/png” as an official Internet media type, joining image/gif and image/jpeg as non-experimental image formats for the Web. Much of this standardization would not have happened nearly as quickly without the tireless efforts of Tom Lane and Glenn Randers-Pehrson, who took over editing duties of the spec from Thomas Boutell.

Current Status

So where are we today? The future is definitely bright for PNG, and the present isn't looking too bad, either. I now have over 125 applications listed (Reference 10) with PNG support either current or planned (mostly current). Among the ones available for Linux are:

- XV: image viewer/converter
- ImageMagick: image viewer/converter
- GRAV: image viewer
- Zgv: image viewer
- xli: image viewer
- XPaint: image editor
- The GIMP: image editor
- Image Alchemy: image converter
- pnmtopng/pngtopnm: image converters
- XEmacs: editor/web browser/operating system/etc.
- gforge: fractal terrain generator
- Fractint: fractal generator
- Ghostscript: PostScript viewer/converter
- GNUplot: plotting program
- PV-WAVE: scientific visualization program
- POV-Ray: ray-tracer
- VRweb: VRML browser
- X Mosaic: web browser

- Arena: web browser
- Chimera: web browser
- Grail: web browser
- Amaya: web browser/editor
- Mapedit: image map editor
- WWWis: HTML IMG sizer
- file(1): Unix file type identifier

Discerning readers will note the conspicuous absence of Netscape Navigator. Netscape is still only “considering” future support of PNG despite the following facts:

1. Netscape was aware of the PNG project from the beginning and unofficially indicated “probable support.”
2. The benefits brought to WWW applications by gamma correction, alpha support and 2D interlacing.
3. The WWW Consortium, of which Netscape is a member, released the PNG spec as its first official Recommendation.
4. Support of PNG is required in VRML 2.0 viewers like Netscape's own Live3D plug-in.
5. Netscape has received considerable pestering by members of the PNG group and the Internet community at large.

Until Netscape either supports PNG natively or gets swept away by Microsoft or someone else, PNG's usefulness as an image format for the Web is considerably diminished.

On the other hand, our friends at Microsoft recognized the benefits of PNG and apparently embraced it wholeheartedly. They have not only made it the native image format of the Office97 application suite but have also repeatedly promised to put it into Internet Explorer. (Theoretically by the time of the 4.0 betas—we'll see if that happens.) Assuming they do, Netscape is almost certain to follow suit. (See? Microsoft **is** good for something!) At that point PNG should enjoy a real burst of WWW interest and usage.

In the meantime, PNG viewing actually is possible with Linux Netscape; it's just not very useful. Rasca Gmelch is working on a Unix plug-in with (among other things) PNG support. Although it's still an alpha version and requires ImageMagick's **convert** utility to function, that's not the problem, Netscape's brain-damaged plug-in architecture is. Plug-ins have no effect on HTML's IMG tag: if there's no native support for the image format and no helper application defined, the image is ignored regardless of whether an installed plug-in supports it. Instead you must use Netscape's EMBED extension. That means

anyone who wants universally viewable web pages loses either way: PNG with IMG doesn't work under Netscape, and PNG with EMBED doesn't work under much of anything except Netscape and MSIE, and then only if the user has installed a working PNG plug-in.

However, support by five or six other Linux web browsers isn't bad, and even mainstream applications like Adobe's Photoshop now do PNG natively. More are showing up every week. Life is good.

The Future

As VRML takes off—which it almost certainly will, especially with the advent of truly cheap, high-performance 3D accelerators—PNG will go along for the ride. JPEG, the other required VRML 2.0 image format, doesn't support transparency. Graphic artists will use PNG as an intermediate format because of its lossless 24-bit (and up) compression, and as a final format because of its ability to store gamma and chromaticity information for platform independence. Once the “big name” browsers support PNG natively, users will adopt it as well—for the 2D interlacing method, the cross-platform gamma correction, and the ability to make anti-aliased balls, buttons, text and other graphic elements that look good on **any** color background. No more “ghosting,” thanks to the alpha-channel support.

Indeed, the only open issue is support for animations and other multi-image applications. In retrospect, the principal failure of the PNG group was its delay in extending PNG to MNG, the “Multi-image Network Graphics” format. As noted earlier, everyone was pretty burned out by May 1995; in fact, it was a full year before serious discussion of MNG resumed. As (bad) luck would have it, October 1995 is when the first Netscape 2.0 betas arrived with animation support, giving the (dying?) GIF format a huge resurgence in popularity.

At the time of this writing (mid-January 1997), the MNG specification has undergone some 31 drafts—almost entirely written by Glenn Randers-Pehrson—and is close to being frozen, although there has been a recent burst of new activity. A couple of special purpose MNG implementations have been written, as well. But MNG is too late for the VRML 2.0 spec, and despite some very compelling features, it may never be perceived as anything more than PNG's response to GIF animations. Time will tell.

At Last...

It's always difficult for an insider to render judgment on a project like PNG; that old forest-versus-trees thing tends to get in the way of objectivity. But it seems to me that the PNG story, like that of Linux, represents the best of the Internet:

international cooperation, rapid development and the production of a “Good Thing” that is not only useful but also freely available for everyone to enjoy.

Acknowledgments

I'd like to thank Jean-loup Gailly for his excellent comp.compression FAQ, which was the source for much of the patent information given above (Reference 11). Thanks also to Mark Adler and the Jet Propulsion Lab (JPL), who have been the fine and generous hosts for the PNG home pages, zlib home pages, Info-ZIP home pages and my own personal home pages. Through no fault of Mark's, that all came to an end as of the new year; oddly enough, JPL has decided that none of it is particularly relevant to planetary research. Go figure.

Greg Roelofs escaped from the University of Chicago with a degree in astrophysics and fled screaming to Silicon Valley, where he now does outrageously cool graphics, 3D and compression stuff for Philips Research. He is a member of Info-ZIP and the PNG group, and he not only maintains web pages for both of those but also for himself and for the Cutest Baby in the Known Universe. He can be reached by e-mail at newt@pobox.com or on the Web at http://pobox.com/~newt/. The Cutest Baby in the Known Universe can be seen at http://pobox.com/~newt/greg_lyra.html. The Info-ZIP home page moved to http://www.cdrom.com/pub/infozip/ at the beginning of the year, and the PNG home page moved to http://www.wco.com/~png/, as noted above.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #36, April 1997

Our experts answer your technical questions.

Doubling Connection Speed

I have heard that it's possible to set up Linux to combine two analog modems into one so as to double the speed of a connection. Is this true? If so, how does this work and where can I get more information on how to do this? I have Slackware 96. —Keith Bell

Load Balancing

What you want to do is called load balancing. There is a feature you must compile into your kernel for load balancing to work and it is designed to work only with SLIP or PPP. The feature you must compile is EQL, or “Serial Line Load Balancing”. As you configure your kernel there is a small amount of help available on the option. If you look at the file **/linux-source-directory/drivers/net/README.eql**, you can get more information on how this works and what you need to do. Be aware that this must be supported by the other end of the connection—either another Linux box compiled with this feature or a Livingston Portmaster 2e. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Mysterious Messages

I am running **named** as a primary DNS server. It appears to be working fine, but my **/var/adm/messages** file is full of lines like the following:

```
Dec 5 09:34:14 lancomm named[105]: NSTATS 849796454 849648847 A=528
PTR=76 MX=96 ANY=202
Dec 5 09:34:14 lancomm named[105]: XSTATS 849796454 849648847 RQ=902
RR=634 RIQ=0 RNXD=49 RFwdQ=393 RFwdR=562 RDupQ=5 RDupR=6 RFail=1 RFErr=0
RErr=0 RTCP=0 RAXFR=0 RLame=15 ROpts=0 SSysQ=53 SAns=509 SFwdQ=393
SFwdR=562 SDupQ=426 SFail=19 SFErr=12 SErr=1 RNotNsQ=886 SNaAns=339
SNXD=49
```

These messages are logged every few minutes. Are these merely extraneous debug messages, or is **named** misconfigured? —Bill Cunningham

Extended Statistics

They are debug messages, and don't mean there is a configuration error. Those messages are the “extended statistics”, a compile-time option for **named**. If you'd like to disable this logging, simply recompile **named** with the **XSTATS** option commented out in the file:

```
~/conf/options.hz
```

—Bob Hauck, Wasatch Communications Group bobh@wasatch.com

More Colors in X-Windows

When I run X-Windows the desktop resolution is 340X400 with 16 colors. I am wondering how to get my X server to run with a resolution of 800X600 with 256 or higher colors. I am having a hard time finding documentation or manual pages on how make this change. I am running Slackware 1.2.1 and using a Cirrus controller. —Matt Linak

Upgrade Needed

Your distribution is very old. You should switch to XFree86-3.2, which includes many more supported cards. Most Cirrus controllers are supported now. Take a look at the **README.Cirrus** file in the XFree86 web site: www.xfree86.org . —Pierre Ficheux, Lectra Systemes pierre@rd.lectra.fr

Setting Up An X Terminal

I am running Linux 2.0.0 and have a second PC that I use a a terminal (serial) using a DOS term program. It's a 486 that used to be my main machine until I upgraded. I have been trying to find information on setting it up as an X terminal, but all the HOWTO and **/usr/doc** files seem to focus on other things. It's my understanding that if I put a small Linux kernel on it and use NFS for root that I should be able to do this as the machine has very limited resources these days. I know I can switch to PLIP for reasonable speed, and I have good documentation on using NFS as root, but I have not uncovered the missing information on setting it up as an X terminal. Can you direct me to a source? —Josh

Inexpensive Hardware and xdm

You do need at least some disk resources to be able to set your H86 up as an X-terminal. There are ways to do a complete net-boot on a PC, but those include

obtaining a 3C509 or NE2000 Ethernet card and a boot ROM. I haven't dealt with this method, though, because hard disks are becoming very cheap.

I recommend getting a 120MB IDE drive (you should be able to find a used one for around $25), and installing that. Then install a minimal Linux system including X, and you are set. You will need networking of some type since most Linux distributions require Ethernet for a network install. If you don't have a CD-ROM on that box, you'll probably want to do a network install, so pick up a cheap networking card (new NE2000 clones run about $25).

Now, for using the 486 as an X terminal, the easiest way is with xdm. You run it on your main machine, configure X on your 486, and you can then run **X -query hostname** on your other machine. That solution will run an X server locally, but will run all binaries off your main machine. —Donnie Barnes, Red Hat Software redhat@redhat.com

Managing Modules

When I build a kernel (2.0.26 is the latest) with loadable module support enabled, I have troubles with the old modules compiled for 2.0.0. When I **make** the modules for 2.0.26, only one module is built and put under **/lib/modules/2.0.26**. How do I manage the other modules? When I put the 2.0.0 modules under 2.0.26, the system complains **you must recompile**. How do I recompile them? —Ivo Naninck

Patch or Scratch?

You did not mention whether you have applied a patch or installed a **linux-2.0.26.tar.gz**. If you have installed from scratch, don't forget to run **make menuconfig**, in order to select which features you want as modules. If you have applied a patch, use:

```
make dep
make clean
make zImage<\n>
make modules
make modules_install
```

*I would prefer using **make zlilo** rather than **make zImage**, but the latter would work. This should compile your kernel and all the modules you have specified. Once the kernel is installed, you should be able to take care of dependencies with the command **depmod -a** the next time you boot. —Mario de Mello Bittencourt NetoWebSlave System Administrator*

Making Rescue Disks

I am wondering if there is a guide on how to make a rescue disk that includes my choice of kernel and root, including some basic tools to help me restore my box in case of an accident. —Eskinder Mesfin

Bootdisk HOWTO

The Linux Bootdisk HOWTO by Graham Chapman (grahamc@zeta.org.au) describes how to create maintenance disks. The text is available at http://sunsite.unc.edu/LDP/HOWTO and ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO. —Martin Michlmayer tbm@cyrius.com

Linux Won't Accept Mask

We have a complex 10B2 network with 4 subnets. We recently got connected to the Internet with ISDN and a Pipeline 50 and were given a single class C address. We have created a subnet scheme for assigning IPs to all of our WFW3.11, Win95, and WinNT computers. When we tried to set up the Linux box as an HTTP and ftp server, we were unable to get the Linux software to accept 255.255.255.224 as the subnet mask. Will Linux do subnets this way? —Richard C. Guglomo

Linux And Subnets

Linux does do subnets, and 224 is a valid mask. Valid masks must have contiguous high bits set and 224 is 1110 0000 in binary. In theory this should work, but there are some pitfalls.

It could be that the IP you are assigning to the interface isn't on a valid 224 subnet. With that mask your network addresses will be 5 bits. In other words, subnets will fall on multiples of 32 (decimal)—0, 32, 64. Those IP addresses are “network addresses” and can't be used for an interface. Similarly, the “all ones” addresses are reserved for broadcast—that would be 31, 63, and so on in this case.

You cannot assign a network address as the IP for a specific interface. So, you can't use something like 192.168.1.32 as the interface address. Instead you should use 33-62 for the devices on the 32 subnet (63 would be the broadcast address and is also reserved), 65-95 for the 62 subnet, and so on. —Bob Hauck, Wasatch Communications Group bobh@wasatch.com

Can't Disable LILO

I have installed Red Hat, and I now want the option of running another OS on my machine, but I have not been able to disable LILO. I have installed DOS, but

when the machine re-boots, I get LILO, and when I press TAB, I have no other options. I have **fdisked** the hard drive through both the Red Hat install program and DOS without any luck. I wanted to move the LILO from the **mbr** but have not been able to do this. —Josef Davis

Adding DOS option to LILO

You can replace LILO with the DOS boot loader by issuing the DOS command **fdisk /mbr**. In your case, however, the solution is to add DOS as an option to LILO. You can do this by adding the following directive at the end of the LILO configuration file, **/etc/lilo.conf**:

```
other = /dev/sda4
label = DOS
table = /dev/sda
```

*You have to replace the value of **other** with the device of your DOS partition; the same applies to **table** where you have to insert the device of your hard disk (**/dev/hda** in the case of the first IDE hard disk).*

After inserting these lines you have to refresh the boot record by issuing **/sbin/lilo** as root. When booting your machine the next time, you will have the option **DOS** within LILO. —Martin Michlmayer tbm@cyrius.com

Archive Index Issue Table of Contents

Advanced search