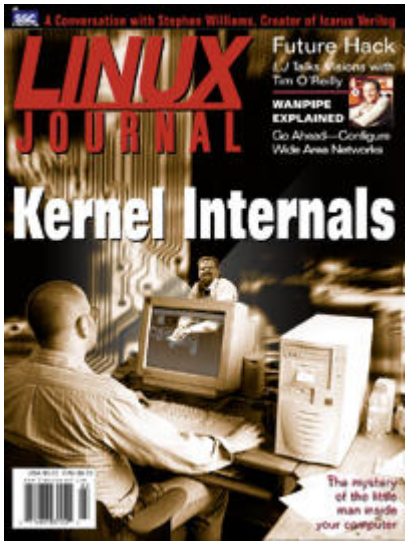
Linux Journal Issue #82/February 2001



Features

Focus: Kernel Internals *by Don Marti*

Making Inodes Behave *by Clay J. Claiborne, Jr.*
Claiborne describes the difficulties he encountered while building Linux systems for General Dynamics.

Journaling with ReisersFS *by Chris Mason*
Mason gives a tour through the Reiser File System: its features and construction.

The Linux Telephony Kernel API *by Greg Herlein*
Herlein explains the integration of the telephony device driver into the Linux kernel.

Inner Workings of WANPIPE *by Nenad Corbic and David Mandelstam*
Corbic and Mandelstam discuss the structure and user interfaces to the WANPIPE drivers as they have evolved and currently exist.

Indepth

Web Servers and Dynamic Content *by Dan Teodor*
Using legacy languages like C and Fortran can aid computationally complex web applications.

Porting from IRIX to Linux *by George Koharchik and Brian Roberts*
Coding for portability to Linux: examples from the ACRT land vehicle port.

Expanding Options for Clustering *by Ken Dove*
The role of Linux in the future of clustering.

Toolbox

Columns

Reviews

Departments

Archive Index

Advanced search

Focus: Kernel Internals

Don Marti

Issue #82, February 2001

It's Special Kernel Issue time.

Expecting a thick magazine called *Linux Journal* to be all about Linux is like expecting a thick magazine called *Martha Stewart Living* to be all about Martha Stewart. No, wait, bad example. But you get the idea. Linux, which is a kernel written to comply with the POSIX standard, is a commodity piece of software. (By the way, *LJ* is two-thirds the size of *Martha Stewart Living* with one-tenth the staff. We don't have to cook a bunch of new recipes, but on the other hand, Martha doesn't have to lay out %{$_} Perl code.)

So, since Linux itself just sits there and works—how boring—we generally fill out the magazine with articles on web servers, development tools and other fun stuff that runs on top of Linux—and, not surprisingly, on other commodity POSIX-compliant kernels too. We could cut out “Kernel Korner” and a few other things here and there, and sell the same content as “POSIX Journal”.

But just because something is a commodity doesn't mean that it's not newsworthy. After all, pork bellies are a commodity, and they make the *Farm Report* every day. So it's Special Kernel Issue time.

Clay Claiborne, a Linux consultant in Los Angeles, had some Digital UNIX (or whatever they're calling it these days) drives that a client wanted Linux to read. It wasn't as simple as just compiling in support for UFS, but thanks to freely available code and documentation, and the developers who know how to apply it, it wasn't impossible either. His step-by-step journey into the process of adding support for a new/old file system is on page 94. Even if you just use ext2fs, it's a great read to help you understand the free software troubleshooting process.

You might have heard that the SuSE distribution now includes Linux with ReiserFS support. We like ReiserFS so far because its journaling feature means that you can just turn your Linux box off and reboot without a time-consuming fsck. But what really makes ReiserFS different from old-school UNIX file systems and from Linux's standard ext2fs? Chris Mason explains b-trees and just what happens when ReiserFS saves your data from an untimely crash, on page 118.

Where the Internet is concerned, things used to be pretty clear—routers just do routing, and hosts just originate and accept connections. But with the rise of load balancers, firewalls (whatever those are; firewall-mongers use the word to mean whatever they're selling) and network address translation, the world of machines through which Internet traffic passes is becoming a strange bestiary indeed.

Nenad Corbic and David Mandelstam write about WANPIPE, the Linux driver software for sangoma PCI WAN adapters with which you can create your own strange beast of an Internet appliance—a frame-relay box for the remote office, a box that connects straight to the T1 line without a CSU/DSU, whatever you need. There are a lot of security, performance, and other problems to be solved on the Internet and WANPIPE (page 100) is part of your toolkit for attacking them.

What's the first kernel to offer a standard API for voice over IP? Linux, of course. Greg Herlein explains what's behind /dev/phoneN and how software like ohphone can use it to make phone calls anywhere at no charge—except of course your local Internet connection—on page 108. And yes, you can plug a real phone into it.

Finally, what do all these kernel projects have in common? Freedom, of course. Centrally planned proprietary software, our society's biggest example of Stalinism in action, is being replaced with a free-market system. This is good.

Every so often, though, people try to sell proprietary components for a working free system, claiming some performance advantage. But this impolite and destructive practice is worse than just not being “down with the revolution”. If we replace bureaucratic Big Software with a feudal system in which local software warlords each try to gain some advantage over the others, we'll find it difficult or impossible to do the kinds of projects covered in this issue.

I'm not talking philosophy or hypothetical here—do a web search for the name of any non-GPL Linux kernel module and get a bushel of archived list messages asking for help when the proprietary module throws some other module's glasses on the ground, stomps on them, then tells it to get out of town and sets the kernel on fire.

We're in the early stages of replacing top-down Stalinist software with a functioning free society, and thanks to politeness and common sense we're doing a pretty good job of avoiding the detour through feudalism. In the long run, freedom works both morally and economically. This issue's kernel development success stories show just how useful a commodity our favorite kernel can be.

—Don Marti, Technical Editor

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Making Inodes Behave

Clay J. Claiborne,, Jr.

Issue #82, February 2001

Claiborne describes the difficulties he encountered while building Linux systems for General Dynamics.

It started with a phone call. Could we build a Linux box that would mount and read a DEC drive and make the data available to NT workstations via Samba? The answer was “I'll have to get back to you.” Because the caller was General Dynamics (the company that makes most of the US Navy's nuclear subs, among many other things) and we are a for-profit Linux company, I planned to get back to them as quickly as possible.

Cosmos Engineering Company has been building custom computer systems for industry since 1984. In 1996 we focused exclusively on building Linux systems. That same year we introduced Linux on a Disk. The next year we became founding Red Hat Hardware Partners. By the time General Dynamics came to us, we had been solving problems with Linux for a few years. This was right up our alley.

The DEC drives were Quantum 9GB SCSI-2 drives with one OSF partition formatted with a UFS file system. Although I wasn't sure of that at the time, I did know that the Linux kernel hackers had been gobbling up new file formats and partition types faster than Carlie gives out free drink tickets at a *Linux Journal* party. For example, between 2.2.15 and 2.3.99pre9 the number of supported foreign partition types went from three to 15. UFS has been in the kernel since 2.0.XX. But then there are different “flavors” of UFS, aren't there? The number of those has also been growing in the march towards 2.4.0-use-me-1.

So I checked with the latest experimental kernel tree, which at the time was 2.3.99pre9. Yes, there was support for seven different UFS types now, although DEC was not specifically mentioned, and support for the OSF partition had just been added. On the strength of this I got back to John Loeffler of General

Dynamics Electronic Systems with a tentative yes, and, “Would it be possible to get a sample drive?” so that we could give them a definitive answer. One FedEx shipment later, I was building the newest experimental kernel with all of the partition types and file systems enabled that might even think about working.

As I suspected from the beginning, mounting the drive as an OSF partition type with UFS type SUN file system read only *appeared* to work. The sample disk had three files on it: rc.local, hosts and a rather large tar ball—oilpatch.tar. We faxed them the contents of the two short files as they requested, and as a result Cosmos Engineering Company was able to get a contract to build a series of Cosmos 500 Linux Servers customized to the task at hand.

That task was to build Linux Servers that could each support four 9gig SCSI DEC OSF drives and allow network attached NT workstations to read the data. This task was then expanded to include the ability to delete files and directories. Okay, no problem, kernel 2.4.0-test1 has experimental write support for UFS. The hardware itself was nothing special. Each server was a 500MHz Pentium III, built around the ASUS i440BX ATX motherboard with 128MB Ram in a mid-tower ATX case. Red Hat Linux 6.1 was installed on the 18GB IBM 2Ultra SCSI system drive. What made this system special was its ability to read the foreign disk format. That capability called for the creation of a special kernel.

The paperwork involved in doing a deal with a company used to subcontracting parts for a destroyer was, well, impressive. We'd never done a contract that involved so many government regulations. But then again, we'd never done a contract with a separate boilerplate stipulating how cost overruns above $500,000 should be handled.

The first sample disk, and a number of others that followed, quite obviously had dummy data on them. We never knew what the real data was. Whatever it is, there must be a lot of it, and they must have wanted badly to keep the data transfer operation in-house to go the route they did. We did know that the end user was the US government, and it did come out in the course of the work that that meant the military. Later, while I was helping them to troubleshoot some SCSI termination problems, I was told that the SCSI drives were enclosed in canisters that plugged into ruggedized quad bays that “We use in everything.” “Everything” in this case being nuclear subs destroyers and other weapons systems. I also knew that with the persecution of Dr. Lee moving ahead full throttle, and with a notebook hard drive full of nuclear secrets turning up missing at Los Alamos, there was a lot of concern in the government with securing the nation's military data. Maybe that had nothing to do with the implied mass data migration. That is just pure speculation. I don't know what our servers are being used for and furthermore, I don't want to know. In talking

to them I always had the feeling that they could tell me but then they would have to, well, you know.

With the contract signed we started to build and test the first of the servers. The project hit a snag when we discovered that there was a problem copying long files from the DEC drives. Files were being truncated after 96KB. That wouldn't do. Why 96KB? That's what I wanted to know.

Further features of this problem were that long files, for example the kernel tree tar ball, could be copied to the DEC drive and then copied back intact in that session. But if the system was shut down and brought back up, and then the kernel tar ball was copied back from the DEC drive, the results would be of the right length but the contents garbage. I suspect it had to do with how inodes are managed in memory and stored on disk.

This began my long descent into Inode Land, and that is what this story is really about. Previously I'd heard about inodes. I knew they were disk structures related to files. I knew I always needed to have enough free ones. Ever get a “no space on the device” message with 4GB free? No free inodes. Inodes were largely a mystery to me. In the next several weeks I learned more about inodes than I'll ever remember.

Inodes are little data structures that define files. In the ext2 file system an inode is 128 bits long. Conveniently, all the stuff that follows applies to both the Linux ext2 file system and the DEC UFS file system. An inode is a table of information including the file's owner, file permissions, create and modify data, file size and most importantly, the location of the actual file data. Everything, in fact, but the file name. The file name is quite incidental to the file. It's there for our benefit. It's a directory entry. Say you want to hear a song. So you double click The_Train_They_Call_the_City_of_New_Orleans.mp3. That's a directory entry that points to an inode that knows where that music lives and who gets to play it. There could be other directory entries for the same inode. This is what is called a hard link. A file can have more than one name but only one inode. That is why Sun folks say, “The Inode is the File.” Every file, including devices, directories and soft links, requires an inode.

Inodes are a limited resource. When you create a file system, you determine the number of inodes you create, and that's that. If you run out of inodes because you put lots of tiny files on a partition with a small inode table, you are SOL even if you have gigibytes free. You may be able to fool a system that has run out of disk space on a particular partition with a symbolic link to a subdirectory on another partition. But you won't be able to fool one that has run out of free inodes, at least not until you delete something and free up an

inode. Soft links, being files that point to other directory entries, require an inode.

Inodes are data structures, but they are usually stored on a disk and read into memory for reference and modification.

Looking at the inode data structure it's easy to see why 96KB is important. The first block of data in the inode has to do with time, date and file ownership, etc. Then, at offset 0x28, we get to what I think of as the “good stuff”--the location of the file's data on the disk. Oh, did I mention that there are different types of inode structures for different types of files, and we are currently discussing only the structure of inodes for DEC ufs regular files?

In the ext2 or DEC UFS 32-bit file system data structure, a four-byte word points to one of many possible data blocks in the file system. So the first 48-bytes of this section consists of 12 four byte pointers to the first 12 blocks of the files data. In the case of this file system, each data block is 8KB long. These are called the direct blocks, because their address can be stored directly in the inode. This also means that small files, 96KB or smaller, can be accessed faster. For larger files the 13th address block in the inode points not to another 8KB block of data, but to an 8KB block of addresses, or 2048 addresses of other 8KB data blocks. This is called an indirect block. For even larger files the 14th address block in the inode points to an 8KB block with 2048 addresses of 8KB blocks, each of which has 2048 addresses to data. This is called a double indirect block. This allows for some very big files indeed.

My problem was that Linux could only read the direct blocks of the DEC file's data. Attempts to read files that required the use of indirect block addressing failed with an “attempt to access beyond end of device” error.

Two things were required to fix this problem: 1) a clear understanding of the structure of the data on the disk, most importantly the inode structure, and how the address information is stored; and 2) a clear understanding of what the operating system is doing with the data it finds on the disk, most importantly how it reads an inode. I had neither of these.

However, I did have the help of the Linux community. Early on I told members of my user group, Linux Users Los Angeles, about the work we were doing and the challenge we faced. A number of people offered insight and suggestions. Christopher Smith went much further. He sent me an e-mail, “If you want to drop off one of these drives and a SCSI controller, I'll try futzing with it during my copious amounts of free time.” I brought him over a prototype server and a copy of one of the sample drives we had received from General Dynamics.

His assistance helped immeasurably in clarifying the problem and how to approach it. Dan Kegel suggested I post to the linux-kernel list. This gave me the courage to do so. He also found the linux-fsdevel list for me. I got a lot of help from people on both those lists.

Peter Swain wrote, it “looks like your indirect block handling is twisted, either through endianness or 64bitness issues. DEC might well have a nonconforming implementation there, but you should conform to it, at least as a mount option.” Jim Nance also thought, “It sounds like some sort of 32/64 bit problem, or perhaps a byte ordering problem of some sort.” He also suggested we investigate with the user-mode port of the kernel and sent us information on how to get it. Peter Rival of Tru64 QMG Performance Engineering suggested we contact Marcus Barrow of Mission Critical Linux, whom he described as “the UFS engineer du jour here until he went to MCL”.

Marcus Barrow responded to my e-mail by saying, “I'd be happy to take a look at this.” He warned “Avoid writing to all three of your disks with Linux. Your file system might be becoming corrupted.” We were already using cloned drives. He wrote extensively on the problem, saying

> Firstly I thought the problem might be dealing with the 8K/1K block/fragment issues. On second thought, I'm more confused. I don't know why you could read the direct blocks but not the indirect blocks. Particularly when Linux can read its own indirect blocks.

Lyle Seaman pointed out that, “Your task would be much simpler if you had the relevant files from /usr/include/fs/* for that OS....They must be in somebody's museum somewhere.” The support we got from Linux users and developers on these two lists was critical to getting this job done.

We also had the kernel source code, which is precisely what made this trip possible. Free software means knowing what the system is doing. It also means being able to modify it. I printed out a ream of files like linux/fs/ufs/inode.c and known associates, and hit the books. Along the way I found some very good material in the Open Source community explaining file systems and the software that loves them. Two tutorials on the Linux file system I'll mention here are

- “Design and Implementation of the Second Extended File System” by Rémy Card, Laboratoire MASI—Institut Blaise Pascal, e-mail: card@masi.ibp.fr; Theodore Ts'o, Massachusetts Institute of Technology, e-mail: tytso@mit.edu; and Stephen Tweedie, University of Edinburgh, e-mail: sct@dcs.ed.ac.uk khg.redhat.com/HyperNews/get/fs/ext2intro.html.

- “The Linux Kernel” by David A. Rusling (david.rusling@arm.com), Chapter 9: The File System, www.linuxdoc.org/LDP/tlk/tlk.html.

So at least in theory I have an understanding of what the OS is doing because I had the source code I was running and the means to modify it and create new binaries. Ordinary stuff for a Linux system.

To fulfill understanding of how the file system is structured, I had to be able to read the data on the disk and understand how it was organized. I needed to see how the inode on the target drive was really structured. I needed to read the inode and see if, and how, the indirect block pointed to the file's missing data.

By then inodes had become very real to me. I devised a plan to trap one, namely the inode for the misbehaving oilpatch.tar. Because I could mount the partition and then remount it read/write with the experimental write code in the kernel, I could change the ownership of oilpatch.tar. So I did.

I knew my area of interest would be near the beginning of the drive, so I copied that area to a file with a command like:

```
dd if=/dev/sda of=root-patch bs=1k count=100000
```

Then I mounted the DEC partition and changed the owner of oilpatch.tar from root (ID 0) to nobody (ID 99), unmounted it and made another file with the command:

```
dd if=/dev/sda of=nobody-patch bs=1k count=100000
```

Taking the difference of these two files reveals a byte value change from 0 to 99 at a certain offset in the files. The four-byte file owner ID # is known to be at a certain offset into the inode, so plugging the proper numbers into a dd command like:

```
dd if=/dev/sda of=copy_of_inode_for_oilpatch.tar
  bs=128 count=1 skip=offset_from_front
```

writes the inode to a file. Now I could examine the inode in some detail. I could print it out, read its hidden messages and find the files data. Fortunately, oilpatch.tar turned out to be a text file of predictable construction. Blocks of text fit together like parts of a jigsaw puzzle, and it's pretty apparent when you have the right order.

What I found was that the first 12 pointers pointed to the first 12 blocks of the file, and the first four bytes of the block pointed to the 13th pointer which pointed at the 13th block of the file. The next four bytes of that block of pointers pointed to the 14th block of the files data and so on.

And there's nothing strange about how it worked at all. No weird “big-endian, little-endian” problems. No unexpected bit shifts at all. All very neat, very straightforward. The way I would have done it. The same way Linux does it, in fact, in the ext2 file system. And the same way the kernel UFS code was expecting it to be, which didn't help to explain why it wasn't working.

I can spend a lot of time looking for a problem where it isn't. Slowly, I came to the conclusion that the problem wasn't in the kernel UFS code at all. The inode was being read as it should. The DEC UFS should read as the ufs_type=sun. In fact, the UFS code had been around for some time, so it was hard to believe it was broken. The problem lay at a deeper level: in the communication between the Linux virtual file system (VFS)--in the by then 2.4.0-test1 kernel and the UFS code, caused by changes in the upper-level code. The pointer to the first indirect block was not being passed properly to the VFS, and this broke the UFS code.

The test of this thesis also became the solution to my problem. I was using first the 2.3.99, and later, the 2.4.-test series kernel because it had support for the OSF partition lacking in the 2.2 production series kernels. If I was right, and the UFS code had only recently been broken, the 2.2 kernel should be able to read the DEC drive with no problem provided it could also handle the partition type.

Well, the back port of the OSF partition code from the 2.4 kernel source to 2.2.16 was easy, even for me, and the resulting 2.2.16 kernel did everything right with the DEC OSF drives, including read and write big files all the livelong day. Since 2.2.16 is a better choice in a production environment, and I had a working solution for this job, we were able to move ahead and complete the work.

We started shipping machines with the hacked 2.2.16 kernel to General Dynamics, and its client, and they are doing whatever they are doing with them. I reported this bug in the 2.4.0-test1 kernel to the linux-kernel and linux-fsdevel lists. I said, “Unfortunately I've found a bug.” Alan Cox wrote back, “Uncovering bugs is good news. The chances are, this one wouldn't have been found before 2.4.0 otherwise.”

That was pretty much the story. Later, when the first systems got on-site, there was a problem with the Windows 2000 workstations mangling long file names, but the Samba people have stayed on top of Microsoft's latest tricks, so an upgrade to the newest Samba release solved that problem.

They also needed to have a static relationship between DEC drives with specific SCSI IDs and mount points regardless of the number of DEC drives installed.

Since Linux likes to assign SCSI hard drive devices sda, sdb, sdc and in the order it finds them, a dynamic approach to drive mounting had to be created.

The original Tekram controllers weren't entirely happy driving the external ruggedized qual bays and got replaced by Adaptec.

Those are the normal hiccups in a job like this. The real task here had been making those inodes behave.

Clay J. Claiborne, Jr. (cjc@linuxbeach.net) is CEO of Cosmos Engineering Company. He has worked in the computer industry off and on for 30 years. He has been a Linux enthusiast since 1995. In 1996 he developed the concept of selling Linux pre-installed on a hard drive and produced Linux on a Disk. He also teaches Linux at Los Angeles City College and is president of Linux Users Los Angeles (LULA).

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Journaling with ReiserFS

Chris Mason

Issue #82, February 2001

Mason gives a tour through the Reiser File System: its features and construction.

There are a few new file systems coming out for Linux these days, bringing some badly needed features for both servers and desktops. I'll briefly describe some of the key ReiserFS features and discuss some details of the journal layer.

ReiserFS stores all file system objects in a single B*tree. The tree supports:

- Dynamic inode allocation
- Compact, indexed directories
- Resizable items
- 60-bit offsets

There are four basic types of items in the tree. Stat data, directory items, indirect items and direct items. Items are found by searching for a key, where the key contains an ID, the offset in the object you are looking for and the item type.

The ReiserFS directories grow and shrink as their contents change. A hash of the file name is used to keep an entry's offset in the directory constant. The tree indexing of this hash allows for very large directories without much performance loss and still provides clean support for NFS and the standard directory operations.

For files, indirect items point to data blocks, and direct items contain packed file data. This packed file data is stored directly in the tree and can share space in the tree nodes with items from other objects. So, for large files, ReiserFS stores block pointers similar to the ones ext2 uses, but for small files we pack the data together to prevent wasted space.

All of these items are resizable by rebalancing the tree. We can append to the packed file data, or if we need another field in the stat data, it can grow to accommodate the new information. The disk format deserves much more detail than I'm giving it here, and you can learn more from the papers on the ReiserFS home page (see Resources).

Large File Support

ReiserFS actually has two main disk formats. The new format introduced in our 2.4 code allows 60-bit file offsets, and the format used in our 2.2 code uses 32-bit offsets. When you mount an older file system under the new kernel, the old format is preserved and large files are not allowed.

There is a mount option for converting to the new format, but the code for mounting the new format under 2.2 kernels is still in beta. Instead of putting out-of-date information in this article, I suggest going to the ReiserFS web site for details on enabling large file support.

How Does the Journal Work?

My goal here isn't to describe APIs or log data structures; I'm hoping to list the major issues involved in file system logging and how ReiserFS deals with them.

Before we talk about how logging works, let's discuss the problem we are trying to solve. In order to have a consistent file system after a crash, updates need to be atomic. They need to happen completely or not at all. For example, to append blocks onto a file, you need to update the file's block pointers, allocate blocks from the free list and update the superblock. If the system crashes in the middle of these changes, the file might have a pointer to a block still on the free list, or the superblock might not have updated stats in it, or the block you allocated could be lost (not in the file and not on the free list).

The ReiserFS journal uses a simple metadata-only, write-ahead logging scheme. The idea is that before any changes are written to disk, they are first committed to a log. After a crash, committed transactions are replayed, which is nothing more than copying blocks from the log into the main disk area.

Writing changes to the log isn't what makes logging complicated. The hard part is keeping the log from slowing your file system down to a crawl. The most obvious optimization is to write to the log in big sequential chunks and lower the number of commit blocks written. Most operations update a small number of blocks, so the journal combines multiple operations into a large atomic unit.

Modified buffers cannot be flushed until they have been copied to the log, and they cannot be freed until they have been flushed. Larger transactions pin

more kernel memory but also make many other optimizations possible. Since ReiserFS stores everything in a balanced tree, the tree frequently needs balancing. Tree blocks are allocated, modified and then freed in another balance later on. With larger transactions, we increase the chance the block will be freed before it is written to the log or the main disk.

It is common for blocks to be logged over and over again. If the superblock is included in transactions one, two and three, it needs to be written to the log once for each transaction. But, it doesn't need to be written the main disk area until after transaction three has finished. The total number of writes needed is lower, and most of the writes are to the sequential log. In some cases, this actually makes logging faster than the original file system was.

Whenever possible, log I/O is done by a worker thread, **kreiserfsd**. This allows log commits to happen in the background, without slowing down user processes. However, the log is a fixed size, so user processes might have to wait for log space to become available before they can start a new transaction. A great deal of care must be taken to make sure processes waiting on the log don't have resources needed by a process already inside a transaction.

Most of the file system does not need to be aware there is a journal layer keeping things safe, but there are a few new rules that need to be followed. First, it isn't safe to modify a dirty buffer. On SMP systems, another CPU might be writing the buffer while you are changing it, which means the modifications would get to disk before the transaction is committed.

Most operations will alter a limited number of buffers, but file writes and truncates are effectively unbounded. Instead of adding the complexity of unbounded transaction size in the journal layer, I chose to code consistency points into these operations. If the current transaction needs to end, they log enough information to make the file system consistent, and then start a new transaction. When data logging is used, **fsync** needs to do the same checks.

Another new rule required by the journal layer has to do with reusing blocks when metdata-only logging is used. Picture these two transactions:

```
1. allocate block 200, insert into the tree<\n>
change and log block 200
free block 200
close and commit transaction 1
2. allocate block 200 as a data block
change block 200, fsync to disk
close and commit transaction 2
[system crash]
```

After the crash, the transactions are replayed in order. While replaying transaction one, the logged version of block 200 is copied into the main disk, and after replaying transaction two, block 200 is a data block in a file. But, the

contents written to block 200 by the fsync are no longer there. ReiserFS avoids this by never allocating a data block until there is no chance a log replay will overwrite the contents with old information. When the file system is full, this means we have to flush transactions to disk and find reusable blocks. Similar checks need to happen if a data block is logged and then written directly later on.

Now that **fsck** isn't needed after every crash, we need to be more careful with lost files. An unlinked file isn't actually deleted until the last open process using it finishes. If the system crashes before the delete operation is complete, the journal will give a consistent file system, but some space will still be allocated to the file. Since the file isn't in the directory tree, there isn't a way to reclaim the blocks.

The easiest way to fix this in ReiserFS is to link the file into a special directory. ReiserFS directories are very fast, and there isn't much locking involved if you aren't worried about file name conflicts. After a crash, the directory is read, and file deletion is finished for any objects left. The special directory doesn't actually need file names at all, just the key information for looking up the file. This fix is not yet integrated into the official ReiserFS releases, but it should be soon.

User Space Transactions

From time to time, people ask for a version of the transaction API exported to user space. The ReiserFS journal layer was designed to support finite operations that usually complete very quickly, and it would not be a good fit for a general transaction subsystem. It might be a good idea to provide atomic writes to user space, however, and give them more control over grouping operations together. That way an application could request for a 64K file to be created in a certain directory and treat it like an atomic operation. Very little planning has happened in this area thus far.

VM Integration

As memory gets low on the system, the kernel needs to start flushing dirty data to disk so the pages can be freed. But, pinned buffers from uncommitted transactions can't be freed until the transaction commits, leaving the VM unable to do anything without help from the file system. We also want to make sure the journal does not use too high a percentage of the system memory for pinned buffers.

We will be working with the VM developers to give memory pressure to the file systems properly. The API is not set in stone yet, but people seem to be leaning toward a flush callback associated with the page, and a generic memory

pressure registration system. It isn't known yet how much of that will happen in the 2.4 kernel and what will be left for 2.5.

ReiserFS and LVM

LVM adds a bunch of cool new features to Linux, one of which is the ability to make read-only snapshots of a device. The snapshot is created very quickly, and copy-on-write is used to keep the snapshot unchanged as the original device is modified. This allows for on-line, consistent backups of most software on just about any file system.

But, the journaled file systems make this a little harder. When sync is called on ReiserFS, we just commit metadata changes to the log, knowing that a replay will make things consistent after a crash. For a read-only LVM snapshot, log replay is not an option. Instead, we can provide a few new generic calls to flush everything to the main disk and pause new file system modifications. While things are paused, LVM initializes the snapshot, so it will be consistent without a log replay. Once LVM is done, the file system is unlocked and writes proceed normally.

Since all the file system operations need to be able to wait on the log, this was easily coded in ReiserFS. LVM 0.9 and ReiserFS 3.6.18 have this functionality, but we are not sure when the generic calls are going to be added in the kernel. Regardless, patches to provide the missing pieces will be available on the ReiserFS and LVM web sites.

Another LVM feature is the ability to relocate extents from one device to another. If you discover an area of the disk is getting higher-than-average traffic, you can relocate those blocks to a faster device. In fact, you could relocate the entire log area to a faster device, reducing head contention and drive seeks. Relocating the log to a solid-state disk could really improve performance on log intensive applications.

Software RAID

In the 2.2 kernels, the software RAID code could write pinned buffers to disk, which breaks the write ordering constraints used to keep things consistent. Only drive striping and concatenation were completely safe, and mirroring was safe as long as you did not use the on-line rebuild code. In the 2.4 kernels, all software RAID levels should work properly with the journaled file systems.

ReiserFS and NFS

ReiserFS has problems supporting NFS because 64 bits of information are required to find an object in the tree, and NFS expects to find an inode with just

the inode number (32-bits long). The good news is the NFS file handle has enough room to store the extra information ReiserFS needs in order to find the file again later, and other kernel developers have written APIs to give the file system control over some of the file handle. By the time this article is out, there should be public patches to add proper NFS support to ReiserFS.

Write Caching

For performance benchmarks, some of the new drives have write-back caching by default. This means the drive reports a write is completed before it is actually on the media. The block is still in the drive's cache, where the writes can be reordered. If this happens, metadata changes might be written before the log commit blocks, leading to corruption if the machine loses power. It is very important to disable write-back caching on both IDE and SCSI drives.

Some hardware RAID controllers provide a battery-backed write-back cache that preserves the cache contents if the system loses power. These should be safe to use, but the cache battery should be checked often. A dramatic performance increase can be seen with these write caches, especially for log intensive applications like mail servers.

Mail Server Optimizations

Mail servers tend to be a worst-case for the journaled file systems because they need to make sure each file operation completes. They use fsync, or a bunch of other tricks to prevent losing messages after a crash, and this forces the file system to close transactions while they are still very small.

Mail servers need a fast way to get new files committed to disk, and data logging can help with this. During the fsync call, you log the data blocks and metadata required to add the file in the tree, producing one large sequential write. If the file being written is a transient spool file, it might never be written back to the main drive. Combined with a fast dedicated logging device, data logging can make a big performance improvement.

As of yet, the ReiserFS 2.4 code does not support data logging. There is data logging code in our releases for the 2.2 kernels, but it needs to be adapted to the 2.4 page cache.

The Competition

The end result of the new Linux file systems should be very interesting. Admins will be able to choose the best product for their applications, and programmers will be able to compare their decisions against the alternatives. Linux as a

whole will benefit as the community picks and chooses the best features from each file system.

Resources

Chris Mason (mason@suse.com) was a system administrator before he started contributing to the ReiserFS Project. He now works full-time for SuSE from his home in Rochester New York.

Archive Index Issue Table of Contents

Advanced search

The Linux Telephony Kernel API

Greg Herlein

Issue #82, February 2001

Herlein explains the integration of the telephony device driver into the Linux kernel.

A year ago, Internet Telephony was a curiosity, and many people thought it would never work for real phone calls. Now, with services like Net2Phone, Deltathree.com and DialPad providing free or extremely low-priced phone calls delivered via the Internet, Voice over IP (VoIP) has reached near-mainstream status. While Linux clients for those services are not yet available, Linux is not being left behind. With the 2.2.14 kernel, Linux has taken a bold lead in the area of computer telephony integration: we have the first modern operating system with a defined kernel-layer application programming interface (API) for telephony support. To sweeten the pot, excellent quality open-source telephony software is already using this API. You can call around the world using Linux and the Internet—and the call is free!

This article will explain the basics of how the telephony device drivers are integrated into the kernel in a way suitable for creating a common API across vendors. Then we'll discuss the basic ideas behind the design and function of the API, and how data and event information are dealt with separately. Finally, we'll discuss how telephony events (like ringing or picking up a handset) are handled in a process called “asynchronous event notification”.

New Kernel Option: Telephony Support

There are many people who've asked why we needed a new telephony support API for the kernel. Even Alan Cox had to be convinced! After all, most of the Internet Telephony software out there can use a sound card, and if that card supports full-duplex (simultaneous playing and recording) and the user has a decent microphone-speaker headset, the quality of the call can be adequate. Why add complexity and a new API?

The answer is quite simple: sound cards are not telephone devices! Sound cards can't generate dial tone, rings, wink, flash-hook or caller-ID—all of which are needed to operate a normal phone device properly. Yes, sound cards are functional for limited telephony use with a headset and microphone, but real telephony cards let you plug that nice cordless phone in and be free from the short cord to your computer...or plug into a sophisticated business phone system. Sound cards also have no chance of providing the ability to plug into the phone line provided by your local telephone company, therefore, losing the ability to do inbound and outbound call control, hop-on and hop-off toll bypass applications or use any of the other new generation telephony software applications being written today.

Also, for any real VoIP use over the Internet, it is necessary to compress the audio data. To be compatible with other VoIP applications and equipment you must support the commonly accepted compression codecs like G.723.1 or G. 729a. Unfortunately, if you want to do this in software you have to license the codecs, and this can cost in the six-figure range (easily!) these libraries are most certainly *not* open source. Telephony cards (like the Quicknet cards mentioned below) have pre-licensed these codecs and provide them built into the hardware. This avoids the mess of paying per-copy royalties for the codecs (it's part of the hardware cost) and lets you develop code with your own choice of license, not one imposed on you by the codec developer. In other words, you can do open-source VoIP software and still use the advanced codecs! These codecs are most definitely not something you would find on a sound card.

Additionally, sound cards can't interface with phones or the phone system, and they can't support hardware-based audio compression codecs. Sound cards are simply different from phone cards.

To be fair, telephony cards are not sound cards either. Sound cards have audio capabilities that far exceed what telephones can do. For example, sound cards are stereo; phones are mono. Sound cards can sample and playback sounds at music frequencies (20Hz-20kHz); phones are limited to voice frequencies (300Hz-4kHz typically). Sound cards have advanced music capabilites and often support MIDI and extensive sound effects. Phone cards cannot do any of that. The hardware and functionality are very different. The device drivers and the API need to be different, too.

But wait, you say, “what about all the great software out there that can work with sound cards, like voice recognition and text-to-speech processing?” Wouldn't it be nice to be able to use that software on phone devices with a minimum of effort? You can. The API for Linux telephony was designed to work in a way that does not preclude the use of software originally intended for a sound card with a telephony device. Yes, some minor code changes will be

needed, but it's not all that hard. I'll explain more about that towards the end of the article.

The push for this new API was spearheaded by Quicknet Technologies, Inc., which manufactures a line of phone cards. In November of 1999, Ed Okerson of Quicknet and myself (I was an employee of Quicknet at the time) proposed the precursor to today's API to Alan Cox. After several weeks of intense e-mail and a pile of code by Ed and Alan, the Linux telephony API was born.

So, how does it work? Let's get started.

The Basics: Phone Devices

At the operating system level, all devices are referred to by numbers. We look in **/dev** and see a collection of file names, but down deep Linux looks at devices based on a major and a minor number. Devices of a particular type all share a single major number; individual devices of that type must each have their own minor number. For example, if you do **ls -al /dev/ttyS0** you'll see:

```
gherlein@tux:~/lj > ls -al /dev/ttyS0
crwxrwxr-x   1 root     uucp       4,  64 Oct 27 06:23 /dev/ttyS0
```

Note that that the file permissions mask has the first character as a “c”, indicating that it's a character device. It's owned by root and is in the uucp group. The next two numbers are not file sizes, like you'd see on a normal file; they are the major and minor number. In this case ttyS0 has a major number of 4 and a minor number of 64.

The Linux Telephony API assigns the major device number of 100 to phone-type devices. Your distribution of Linux has probably not created the devices for you, like it has for **/dev/ttyS***, **/dev/audio** and other older, commonly accepted device mappings. If **/dev/phone*** devices are not present on your system, you'll need to make them before using the Linux Telephony API. You can fix this yourself quickly with the following command (as root):

```
mknod /dev/phone0 c 100 0
```

This creates a new device file called **/dev/phone0**. It's a character device with a major number of 100 and a minor number of 0. Refer to the **mknod** man page for more details on this command. You only actually need enough device files for your hardware, but most folks create devices 0 15 by default.

Note that there is presently an error in the file **/usr/src/linux/Documentation/devices.txt**. This file, which provides official assignments for all major numbers it currently states that Linux telephony is to use major number 159 and that 100 is deprecated. This is an acknowledged error and will be fixed in future

documentation. The correct major number for Linux telephony (and the number used in the kernel) is 100.

The Phonedev Module—Device Brokering

Alan Cox developed the phonedev module based on a similar approach that he took for the Video4Linux Project. There will be many vendors who create products capable of being a phone device. Rather than having multiple telephony product vendors, each requiring their own major number, they can all use major number 100 and the commonly defined **/dev/phoneN** (where N is the device number).

All of these must present a common basic interface to user-space software, that is, they must all follow the same common API.al, though they may provide extensions particular to their product. Vendors will create their own device driver module that will implement this common API as an external interface by dealing with the inner details of their particular hardware.

The phonedev module solved the problem of mapping the minor device number to a specific vendor-type module at runtime. The source code is in the **files/usr/src/linux/drivers/telephony/phonedev.c** and the header files, phonedev.h and telephony.h, are located in **/usr/include/linux**. Here's how it works.

Listing 1. phone_device Structure

Every phone device must use a phone_device structure (see Listing 1). Likewise, every phone device must call two functions to interact with the phonedev module in order to register and unregister itself. These are defined as:

```
extern int phone_register_device
    (struct phone_device *, int unit);
extern void phone_unregister_device
    (struct phone_device *);
```

At load time, the phonedev module sets itself up and waits to service other telephony devices. When a telephony driver is loaded (by **modprobe** or **insmod**, as discussed later) it calls the phone_register_device function. A simple explanation of this function is that it keeps a pointer within the phonedev module to the phone_device structure, searches for the first open minor number and assigns that to the phone device, and then increments a counter to track the fact that something is using the phonedev module (to prevent it being unloaded while in use).

The practical implications of this are simple: the first telephony modules loaded will be assigned the first available (lowest numbered) minor numbers. This is

crucial to understand in cases where modules from different vendors need to coexist on the same system, and specific assignment of a particular card is desired to match a particular minor number (a specific **/dev/phoneN** device). In other words, if you have a device from XYZ company and a device from ABC company, and you want ABC's card to be **/dev/phone0**, you'll have to make sure that the driver for ABC gets loaded first.

All devices must provide at least the basic functions to interact with the device: open, read, write, close, etc. These are all part of the “file operations structure” (see linux/fs.h for details). Each device defines the functions appropriate for itself.

Anytime a program opens a **/dev/phoneN** device, it's actually calling the **fopen** function defined in the phonedev module's file operations structure. This function performs the following operations:

- It grabs the minor device number from **/dev/phoneN** inode.
- It builds a string of the form **char-major-%d-%d** using the major and minor numbers. In the case of minor number 0 (corresponding to **/dev/phone0**), this string would be **char-major-100-0**.
- It uses this string in a call to **request_module** to ask that the module be loaded. This has the same effect as if the program **modprobe** was called (in fact, it actually starts a separate kernel thread and executes **modprobe** in it). This ensures that, if the device is a module instead of a part of the kernel, **kmod** has a chance to load the module before phonedev tries to use it.
- It then calls the **fopen** function in the phone device module to perform the actual act of opening the individual device.

As you can see, the phonedev module has two basic purposes: to assign minor numbers to phone devices dynamically at load time and provide a clean way to autoload needed phone device modules at run time. It's clear, though, that a good understanding of modprobe and module dependency is needed to fully understand the telephony modules and their interactions.

Modprobe and Phone Devices

If you are using kmod to automatically load kernel modules as you need them, then you will need to make sure that several aliases are defined in the **/etc/modules.conf** file.

First of all, the system needs to know that char-major-100 is the phonedev module. Add this line to define:

```
alias char-major-100   phonedev
```

Now the system will need to know what actual phone device modules to associate with specific minor numbers. As we learned above, this may not guarantee that the phonedev module actually assigned that minor number to the phone device at module load time, but we'll cover that in more detail below. In the following examples we'll use the telephony cards from Quicknet Technologies Inc. (www.quicknet.net). These cards have Linux drivers in the kernel and are relatively inexpensive compared to most telephony cards. Quicknet's device driver is **ixj.o** and the module name is **ixj**. This driver is used for all of their telephony card products (it's smart enough to handle the ISA, PCI or PCMCIA flavors as well as knowing which cards have which kinds of phone interface circuits). To define that Quicknet's driver is associated with **/dev/phone0**, add this line to **/etc/modules.conf**:

```
alias char-major-100-0  ixj
```

As you'll recall from the discussion of phonedev's fopen function above, phonedev will build a string of the form **char-major-%d-%d** and fill in the parameters with 100 (major number) and the requested minor number. In our example, attempting to open **/dev/phone0** would result in phonedev attempting to load **char-major-100-0.** That device is unknown to the kernel module loader. The alias command above **maps** that string to the module name ixj. When we try to open **/dev/phone0** the phondev module will autoload the ixj module for us, and then call the fopen function defined in the ixj module to open the device (assuming your kernel was built to support the kernel module loader with CONFIG_KMOD=y).

The Basic Phone Device API

A fundamental premise of the Linux telephony API is that the only thing read-from and written-to phone devices using **read** and **write** is actual audio data. This is separate from—and handled quite differently from—event type information.

Audio Data

What, specifically, is audio data, and how is it different from event data? Audio data is the result of the analog-to-digital (A/D) conversion (and possibly data compression) on the audio signal present at the microphone of the telephony device. The **offhook** signal resulting from picking up the phone handset is an **event**. The tone that results from the user pressing a digit on the phone handset is an event even though that action may also generate audio. An incoming ringing signal is called an event. In short, all non-microphone inputs are events. All microphone input (and the corresponding speaker output) is audio data. This audio data is read and written from the phone device using the standard read and write system calls.

Most phone devices will provide audio data compression in the device. In fact, for successful Internet telephony applications, some form of audio data compression is required. These compression techniques are called “audio codecs” (or codecs for short), and there are a set of commonly implemented and interoperable codecs. The Linux telephony API includes defined constants for most of these common codecs; however, a particular phone device may or may not support them all. Control of what codec is in use is handled by an **ioctl** system call.

One large difference in how Linux telephony API differs from what used to work with sound cards is that Linux telephony API is “frame-oriented” while sound cards are “byte oriented”. A device that is frame-oriented reads a discrete frame of data corresponding to a unit of time. This is done because all audio codecs operate on a period of audio data at a time (usually 10, 20 or 30 milliseconds' worth of data). Because use of a compressed codec is the norm for network telephony applications, this is the normal and expected mode of operation for phone devices. Sound cards do not have this restriction and are free to read and write a variable number of data bytes on any give call. The API defines “frame sizes” for each codec, and raw uncompressed audio data from the device is one of the codec choices. For example, the LINEAR16 codec (uncompressed 16-bit sound samples) has a default frame size of 240 bytes—corresponding to 30 milliseconds of data at the default 8000Hz sample rate. Every read operation from the device will result in 240 bytes of data or nothing. Of course, you can change this behavior with ioctl calls to adjust the size of a frame for uncompressed codecs.

Controlling Phone Devices—the ioctl System Call

Commands for the phone device to perform certain actions are not written to the phone device using a write call—only audio data is written to the device. To control the phone device, a set of ioctl functions are defined to handle the basic phone activities. This basic set of ioctl functions are defined in **/usr/include/linux/telephony.h**. Vendors who wish to extend that basic set of capabilities may do so, but those functions are limited to their own device driver and are outside the scope of the common Linux telephony API.

An example will best illustrate this potential. Using the telephony API with a Quicknet Internet PhoneJACK card and a phone plugged into the card (called an FXS port or POTS port by telephony folks), Listing 2 shows a brief program to ring the phone.

Listing 2. Ringing the Phone

You'll notice that the device is opened, defaulting to /dev/phone0 if the user does not provide a specific device name on the command line. The maximum

number of rings is set with an ioctl call using the PHONE_MAXRINGS constant. The phone is then instructed to ring using the PHONE_RING ioctl. This example program is a simplified version of the LGPL module **ring.c** found in Quicknet's Software Developer Kit (SDK). It's too simple to do anything more than illustrate the technique and demonstrate the use of ioctls to control a phone device, but real-world programs need not be much more complicated, the API is fairly simple. All the Linux telephony API ioctl constants are defined in the header file **/etc/include/linux/telephony.h**.

It's this basic fact that allows software written for sound cards to be adapted easily for use with phone devices; like sound cards, the only data written using the read and write system calls is audio data. In addition, the low-level constants used in the defined telephony ioctl calls were designed to avoid conflict with existing sound card ioctls. Porting an application that expects a sound card to use a phone card instead will primarily involve handling the errors returned from the sound card ioctl calls your code makes. It should be possible (though perhaps not easy) to write a wrapper that opens the telephony device and spawns a child process (inheriting the open file descriptor to the phone device) that runs software expecting a sound card interface. While it's not totally transparent, it's possible and should not actually be that difficult.

Asynchronous Event Notification

Events that occur on the telephony side of the device need to be communicated to the user-space software running the phone. Old and crude methods for this required the software to poll the device continuously for status and changes. The Linux telephony API avoids that, of course, and provides two different techniques, both generically called “asynchronous event notification”. The first method uses signals for indication, and the second uses the “exception bit” in the file descriptor set for exceptions. I'll cover both techniques in order.

The use of signals for event notification requires three steps: first, prepare and declare a signal handler function for the SIGIO signal; second, set the process ID (PID) for the running process to receive the signal; third, enable the generation of the signal on the open files descriptor using the fcntl system call. An excellent description of these steps can be found in Chapter 12 of *Advanced Programming in the UNIX Environment* by W. Richard Stevens (a worthwhile book to have in any case). Again, a short example will probably clear this up. Assuming you have an open file descriptor ixj1 that is an open phone device, you can enable asynchronous event notification with signals with the following code snip:

```
signal(SIGIO, &getdata);
fcntl(ixj1, F_SETOWN, getpid());
```

```
   oflags1 = fcntl(ixj1, F_GETFL);
   fcntl(ixj1, F_SETFL, oflags1 | FASYNC);
```

and a related signal handling function (**getdata** in the code above) to process the data. When you get the signal, you still do not know what kind of event occurred—only that an event did occur. Your program would then have to make an ioctl call to the phone device to ask what kind of event was detected (more on this below). In addition, if your program has more than one open phone device file descriptor, you will not know which one generated the signal. And, signals can be complicated to deal with and can be unreliable in a multi-threaded program and, so, are avoided by some developers. These factors limit the effectiveness of this method, leading to the more useful case of using the "exception bit" in the file descriptor set for exceptions.

It's common for programs to set read and write exception sets and then use the **select( )** system call to wait for a file descriptor to be readable or writable. A less well-known aspect of the select( ) call is the "exception set". The Linux telephony API uses this exception set to signal a process that a telephony event has occurred. Listing 3 presents a simple example using the file descriptor ixj1.

Listing 3. Using an Exception Set

This extremely simple (and not really useful) example is purely to illustrate the technique of using the exception set and select( ) to detect events. The phone device driver will set the appropriate bit in the exception set if an event has occurred, causing select( ) to return. The user can screen the read, write and exception descriptor sets to determine if its own file descriptor was marked by the device driver as ready for that kind of operation. If data is ready to be read, the statement **FD_ISSET(ixj1,&rfds)** will return **TRUE**; **FD_ISSET(ixj1,&wfds)** statement returns **TRUE** if the device is ready to be written to. And, if a telephony event has occurred, the **FD_ISSET(ixj1,&efds)** will return **TRUE**. So, how do you detect what specific event occurred?

The API provides a special PHONE_EXCEPTION ioctl call and an associated telephony_exception structure to decode the return value. This call will set bits in the structure that indicate which of the telephony events occurred (there may have been several). In the above example "hookstate" bit is examined in the statement **if(ixje.bits.hookstate)**, and if that bit is set it indicates a change of status. An ioctl call is then made to determine if the phone is on or off hook. Real-world code would have used a large select or an extended if-else-if ladder to examine the contents of ixje.bits after the PHONE_EXCEPTION ioctl call. A detailed explanation of how to use this technique is beyond the scope of this article, but please refer to the **/usr/include/linux/telephony.h** file for details of what kinds of events can be detected.

Open-Source Programs Available Now

There are many open-source programs available now that use this API. However, the most well-known and widely used program is **ohphone,** a console application using the open-source OpenH323 library. Ohphone is part of the OpenH323 Project (www.openh323.org) and is in daily use by thousands of people to make free, high-quality phone calls over the Internet. Ohphone not only fully supports the Linux telephony API, but it is also compatible with other H.323-based products like Microsoft NetMeeting<+H>tm<+H> and Cisco voice-enabled routers. A more detailed discussion of this fine software is too much for this article, but you're encouraged to look at its web site to catch the latest news. The company that developed the OpenH323 library was recently acquired by Quicknet Technologies, Inc. as part of Quicknet's efforts to ensure continued major development effort of this open-source project. With such full commercial backing and a commitment to open source, I expect the OpenH323 Project software to become even better in the near future.

Conclusion

The Linux telephony API provides a common and consistent interface for developing telephony software on Linux. While there is currently only one vendor (Quicknet Technologies, Inc.) with fully compliant drivers for this API, several others are working toward compliant drivers. The API is lean, well designed, will not conflict with the existing API for sound cards and provides the ability to support multiple vendors behind the same interface. There's sure to be some exciting new telephony software developed for Linux in the coming year.

Resources

Greg Herlein has been an avid Linux developer since 1994. His company, Herlein Engineering, currently offers Linux/UNIX consulting, especially in the areas of telephony software development. He lives and works in San Francisco, California. He can be reached at greg@herlein.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Inner Workings of WANPIPE

Nenad Corbic

David Mandelstam

Issue #82, February 2001

Corbic and Mandelstam discuss the structure and user interfaces to the WANPIPE drivers as they have evolved and currently exist.

From the start, Linux has been the operating system of choice for network appliances—devices that provide services such as network address translation (NAT), firewalling, virtual private networks (VPN), mail services and web caching. It was therefore a natural requirement, from the earliest days, that Linux provide internal support for wide area network (WAN) connectivity.

In 1994, Sangoma began developing its WANPIPE code and utilities for Linux to supplement drivers that had been written by third parties to support Sangoma WAN cards.

Intelligent Adapters

WANPIPE supports the Sangoma S series. Included in this series are cards such as the S514 PCI and S508 ISA—intelligent adapters that support most WAN protocols in firmware, including Frame Relay, PPP, HDLC, X25 and BiSync.

Because protocols are supported in firmware, design of the device driver is much simpler. It is easier to port to a new operating system as there are fewer chances of failure and CPU load is kept to a minimum. These characteristics enable a relatively slow machine like a 486 to use a Sangoma adapter and Linux to create a powerful T1 router.

With protocols isolated on the board, it is possible to test and certify the protocol implementations of one operating system and know that they will work identically on any other operating system. If necessary, a protocol update can be installed on the fly, without recompiling the driver or the kernel.

Sangoma adapters can have two different physical interfaces, T1 (CSU/DSU on board) or serial V.35/X.21/EIA530/RS232. The card with the T1 interface allows the server to connect directly to the T1 line without an external CSU/DSU.

WANPIPE Device Drivers

Sangoma S514 and S508 cards support up to four high-speed sync lines, each carrying a multichannel WAN protocol. The driver architecture was based on the following requirements:

- Multi-adapter support, where each adapter can have up to four physical links
- Each link can have a maximum of 255 logical channels
- Control and configure each link separately
- Control and configure each logical channel separately
- Support multiple protocols: Frame Relay, CHDLC, PPP, X25, SDLC, etc.
- Easily expandable (future protocols)
- Support for both Routing and API applications simultaneously
- Secure socket for API applications (No silent packet dropping allowed)
- Local and remote debugging support
- SNMP Support
- Fast Tx and Rx paths
- Proc file system support: statistics and debugging
- Driver Message/Event Logging
- Easy updates and upgrades

The following design rules were adopted in developing the driver:

1. WANPIPE drivers map each active physical link to a separate device in the kernel. Each physical line can be configured, restarted or debugged without conflicting with other sync lines. WANPIPE drivers can support up to 16 devices, four cards with four physical links.

2. Since WAN protocols can have many logical channels on a single physical link, each channel is mapped to a network interface. WAN protocols, such as Frame Relay or X.25, support one-to-many connections through the use of logical channels. For each physical line, WANPIPE supports up to 255 logical channels for X.25 and 100 logical channels for Frame Relay. Each logical channel can be restarted or reconfigured without bringing down all other channels on the same physical link.

3. Management/debugging interface calls are user datagram protocol (UDP) based and OS independent. Sangoma adopted a common UDP-based interface for collecting statistics and managing WAN links to provide an alternative to SNMP-based tools that are often complicated and costly. Sangoma felt that users should be able to easily interrogate and manage the system remotely, using tools that are included in the WANPIPE package.

The system developed is UDP-based and operating system independent so that, for instance, a Windows GUI application can be used to monitor a remote Linux system. The system is password-free but can be made to operate in a highly secure manner.

4. Each network interface can support either API or ROUTING mode. Aside from IP routing, many of Sangoma's customers use the Sangoma S series as communication building blocks for their own applications, using our published applications program interfaces (API). These applications include such diverse uses as:

- Providing unidirectional broadcasts of financial information and newscasts over satellite links.
- Monitoring cellular switch information using X.25.
- Emulation of IBM mainframes or controllers over SDLC, X.25 or BSC.
- Controlling military hardware using HDLC LAPB.

For maximum flexibility, the architecture was designed so that API calls and standard IP routing traffic can coexist on any physical port. For instance, a set of X.25 logical channels can be used to provide standard IP connectivity between a series of locations while other channels are used to exchange point-of-sale (POS) credit card swipe information that is not IP-based. The driver can accept API or routing packets simultaneously, depending on the network interface setup.

5. No API packets to be dropped by the stack. IP stacks are designed to silently discard packets when congested. This is accepted behavior in the IP world, where data integrity is ensured by higher-level TCP protocol. However, in the case of error-correcting protocols such as BSC, HDLC LAPB, X.25 and SDLC, it is assumed that once a packet has been acknowledged at the link level, delivery to the application is guaranteed. Therefore, it was absolutely necessary that the WANPIPE raw socket stack would not silently drop packets.

6. WANPIPE device drivers should be loaded as modules into the Linux kernel.

Using modules, rather than compiling the kernel, makes it easy to update the driver. Also, because only the modules need to be compiled, no reboot is needed.

Employing the above design rules, WANPIPE device drivers maximize the utility of the S series cards. Clearly defined data, debugging and (re)configuration paths enable simultaneous multiprotocol operation to be efficient, configurable and manageable. The design of the driver is shown in Figure 1.

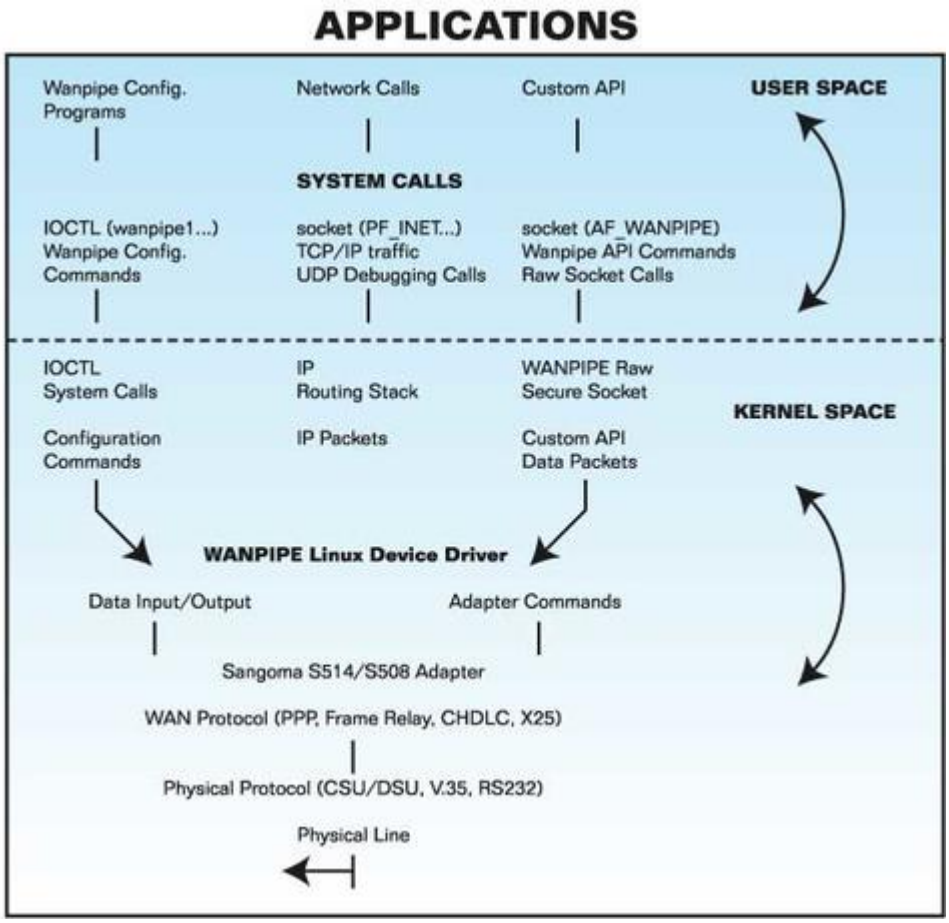


Figure 1. How the WANPIPE Device Driver Fits into the Linux Kernel

WANPIPE and the Linux Kernel

Figure 1 shows how the WANPIPE device driver fits into the Linux kernel architecture. Linux is divided into two operating regions, the user space and the kernel space. All applications, dæmons and utilities execute in the user-space, while kernel and device drivers execute in the kernel space. Communications between user space applications and the kernel are facilitated through system calls such as ioctl.

Device drivers, an integral part of the Linux kernel, interface hardware components to the operating system. Drivers are usually compiled into the

kernel or provided as independent, separate modules that can be dynamically loaded or unloaded at run time.

Sangoma used modular drivers in WANPIPE because modules can easily be updated and reloaded while the kernel is running, eliminating the need for costly system reboots.

WANPIPE, being a network device driver, uses network interfaces to bind to the Linux kernel stack. The network interfaces contains Level 3 protocol information (IP) as well as driver entry points, enabling the Linux kernel stack, via the network interface, to control driver operation: interface shutdown, startup, statistics and data communications.

WANPIPE Configuration

The WANPIPE configuration process starts with creating a detailed configuration file that outlines the hardware, protocol and IP options as well as location of the adapter firmware. Once completed, WANPIPE driver modules are loaded into the kernel. The initial module load allocates necessary resources, initializes and sets up the proc file system directories and enables the ioctl system calls. Since loaded modules do not have enough information to completely configure the card, ioctl system calls are used to pass the contents of the configuration file to the driver. The final step in WANPIPE configuration is to configure and start up network interfaces using the ifconfig() command. The sequence is shown in Table 1.

Table 1. Configuring and Starting up Network Interfaces

WANPIPE and Routing

The kernel IP layer provides a packet transfer service; that is, given a packet complete with addressing information, it will take care of the transfer. In conjunction with the IP layer, the routing table (see Table 2) determines the forwarding order of all incoming packets.

Table 2. Kernel IP Routing Table

Once the WANPIPE network interface (wp1_fr16) is up and running, the kernel routing table is updated with the interface's IP information. The wp1_fr16 interface has two entries. The first one specifies the destination network and the second indicates a default route, meaning that all IP addresses not specified in the above routing table will be forwarded to wp1_fr16 interface.

Upon successful driver configuration, network interfaces and routing tables can be viewed and modified from the user space using the standard Linux commands:

- ifconfig—display or modify network interfaces
- route—display or modify the routing table

WANPIPE and the APIs

An API is used to send and receive custom RAW, non-IP packets to and from the card. Since data is not communicated in IP format, the network interface is configured without the IP address. This effectively removes the kernel routing table entry and unhooks the IP routing stack from the WANPIPE driver. Non-IP communication is achieved using the RAW socket calls to the driver. As the name implies, packets are transferred without any modification.

To ensure that packets that had been acknowledged at the card level were never lost, a secure socket solution was developed: a custom WANPIPE socket that guarantees delivery in both upstream and downstream directions. The WANPIPE socket is based on the Linux RAW socket, developed by Alan Cox and others.

Developing with the WANPIPE Secure Socket: X.25API

We provide the following as an example of working with the WANPIPE API set. We have chosen X.25 as a line protocol because it is probably the most complicated, involving call set up and tear-down, logical channel management and exception condition handling. X.25 is a packet-switched WAN protocol that (generally) uses a public packet-switched network to route packets to different end points. In operation, it appears to be similar to TCP/IP, although the underlying mechanisms are quite different. Line speeds are almost always below 256KBps, usually below 64KBps. Its operation is analogous to a telephone. A call must be initiated, and once the call is accepted, data can be transmitted. When data transmission is over, the call is cleared. Using the WANPIPE secure socket, X.25API programming is very similar to TCP/IP programming.

To continue our example, we assume that the WANPIPE drivers are configured and successfully started, and that the X25 link is up and running (see Listings 1 and 2 at our FTP site—ftp.linuxjournal.com/pub/lj/listings/issue82).

WANPIPE Debugging

WANs, by their nature, are quite complicated. There are usually several players, including one or more Telcos, often a public network provider and often a

separate ISP that adds to the confusion. The inevitable line teething problems and ongoing line debugging can often denigrate into futile finger-pointing exercises.

For this reason, a major part of the WANPIPE development has been devoted to debugging. Sangoma's philosophy is to provide customers with enough debugging information so that the customers can solve most problems by themselves. Furthermore, if support is necessary, Sangoma tech support must have enough information to solve the problem remotely.

The xPipemon Programs

Each WAN protocol has its own debugging utility that is used to determine the status of the driver and physical line, obtain protocol state and statistics, as well as raw and interpreted line traces. The data transfer involved in the monitors is UDP-based. Remote systems can be debugged via the Internet, without logging into the user system, while system security can be tightly managed. UDP calls are OS-independent, meaning that a remote Linux machine can debug a WANPIPE card running in a FreeBSD or Windows machine.

Since system security is an important issue, the UDP debugging commands can be turned off by setting the UDPPORT to 0, or better yet, by setting the TTL (time to live) value to a small one. By setting the TTL value to one, for example, only users that are logged into the machine or located in front of the first router will be able to operate the debugger. The TTL and UDPPORT values are configurable in the WANPIPE configuration file.

A current list of monitors and typical commands is given in Table 3. Under Windows, X and other graphic environments, the complicated command lines are replaced by simple GUI applications.

Table 3. Monitors and Typical Commands

The driver receives management requests via the UDP/IP stack. All received requests are then forwarded into a low priority queue. A low priority thread handles requests and sends the results back up the stack, to the originating IP address. UDP debug requests can also come from the network where the request is sent back through the line. Management connections through the network interface are treated differently from traffic from “above”. Only statistics are available through the network, while access from above allows the user to also reconfigure, test and set up the CSU/DSU and run line traces.

Proc File System

The proc file system is a memory mapped virtual directory structure that is used to provide driver and kernel information. Management systems, such as SNMP, use the proc file systems to obtain kernel/driver statistics and states. The WANPIPE driver binds into the proc file system by setting up /proc/net/wanrouter directory. This directory contains virtual files for each WANPIPE device. WANPIPE configuration and statistics can be obtained by reading/opening supported /proc files. To display tx, rx and error statistics for, say, the wanpipe1 device, use this command: **cat /proc/net/wanrouter/wanpipe1**

Log Messages

All WANPIPE events are logged via the syslog dæmon, in the /var/log/messages file. Note, syslog can be reconfigured to forward messages to a different location. To view the messages log continuously, execute: **tail -f /var/log/messages**.

Nenad Corbic, senior data communications specialist—Heading the Linux development team at Sangoma, Nenad works with senior management to ensure Sangoma's Linux customers are provided with innovative wide area network (WAN) communications technology. Nenad is responsible for WANPIPE device driver design and development, WANPIPE quality assurance, new product development and third-level customer/developer support. He also has interests in the Linux routing project and embedded Linux development. Nenad holds a BEng in Computer Engineering from Ryerson Polytechnic University in Toronto.

David Mandelstam, chief technology officer—Spearheading Sangoma's growth since inception, David remains committed to developing and improving technology for wide area network (WAN) communications. As chief technology officer and founder of Sangoma, David directs the technology strategy and corporate research activities, managing development of new product technologies and overseeing the entire manufacturing cycle. David holds a BSc in Mechanical Engineering from the University of Witwatersrand in South Africa, an MSc in Aerodynamics from the Cranfield Institute of Technology in the United Kingdom and a BComm from the University of South Africa.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Web Servers and Dynamic Content

Dan Teodor

Issue #82, February 2001

Using legacy languages like C and Fortran can aid computationally complex web applications.

When web servers first appeared their primary purpose was to serve up selected information from the machine on which they ran. The idea was to simply take the contents of a file and transmit them over a TCP connection in HTTP format. The inherent limitation discovered early on was that dynamic content could not be delivered, and the CGI interface was defined and added to web servers to address this.

The Common Gateway Interface (CGI) provides a way for the web server to execute a process whose output is determined by the code that it executes, and then take this output and pass it back to the client browser as if it were the contents of a static file. Since that time many variations that combine scripting engines and CGI have evolved that simplify the job of the programmer and make the execution of multiple external threads more efficient (Perl, Python, PHP, etc.). However, for the most part, these scripting languages have the drawback of needing to be interpreted. Also, they overlook the fact that enormous bodies of code already exist in such languages as C and Fortran (remember it?) to solve complex, computationally intensive applications. While building dynamic content based on the results of queries is very useful, it is not realistic to apply image data transformations or calculate Fourier Transforms using these scripting languages, as the time required to complete such calculations is very long with respect to the response time which the user expects.

Therefore, because of the existing code base and the existence of dynamic content that has to be generated by some algorithms that cannot be efficiently implemented in scripting languages, there is a definite need for developing programs that use the CGI, written in a traditional language like C or Fortran and compiled down to native code.

How Web Servers Pass Data to Your Program

There are two ways in which data in the HTTP protocol is passed from the browser to the web server: the GET data (specified as part of the URL, commonly that part that appears after the "?" in the URL) and the POST data (the collected name-value pairs of all of the fields in the form that is being submitted by the web browser). To figure out the GET data, just look at the URL —http://www.mydomain.com/pages/external.cgi?additional-data.

```
GET data portion: additional-data
```

To figure out the POST data look at the source of the document generating the request, and find the form that is being submitted:

```
<FORM>
<INPUT TYPE="text"NAME="fld01"
VALUE="val01">
<INPUT TYPE="hidden"NAME="fld02"
VALUE="val02">
<INPUT TYPE="checkbox"NAME="fld03" CHECKED>
<SELECT NAME="fld04">
<OPTION VALUE="val03"> Val-03
<OPTION SELECTED VALUE="val04"> Val-04
<OPTION VALUE="val05"> Val-05
</SELECT>
</FORM>
                    POST data:
   fld01=val01&fld02=val02&fld03=on&fld04=val04
```

As you can see, POST data is submitted in the form of a continuous string identifying each field/value pair separated by an equal sign ("="). Each field/value entity is separated by an ampersand ("&").

There are actually other complexities involved in the format for passing in POST data. Many characters need to be "escaped" in order not to confuse the web server with control characters or separator breaks. This is solved by inserting plus signs ("+") for spaces and escape sequences of the format "%[0-9,A-F][0-9,A-F]" in the place of non-printable characters, ampersands, pluses and equal signs. (For the purist, yes, it is true that spaces can be represented as either the "+" symbol or as the escape sequence "%20". All versions of Apache and IIS with which I have worked accept both.)

The web browser passes GET and POST data to the external thread by different mechanisms. GET data is placed in an environment variable visible to the context local to that thread. This environment variable is "QUERY_STRING". Therefore, gaining access to GET data in C/C++ is a simple matter of this command:

```
char *pszGetData = getenv("QUERY_STRING");
```

(This should work in all UNIX and in all Microsoft development environments.)

On the other hand, POST data is passed to the external thread on the standard input stream. For those of you not familiar with streams, it is the same data producer as the keyboard, so whatever it is you've been doing to read input from the keyboard, that is the mechanism you will use to access POST data. However, an inherent liability with streams is that you have no way of knowing how much data is waiting for you. The obvious solution is to keep reading byte by byte from that stream until there is no more data and keep resizing a dynamically allocated buffer accordingly. However, web browsers provide the developer with another piece of information that saves them the trouble of having to grow buffers, incur the added overhead and deal with handling the exceptions that occur when one of multiple dynamic memory allocations decides to fail.

When a web browser passes POST data to the standard input of an external thread, it places all of the POST data there in one shot; therefore, there is never any chance that additional data will be added to the stream after you have first encountered a data portion on that stream.

The web browser also tells the process just how much data it has placed in the standard input by writing (as ASCII text) the number of bytes waiting to be read from the standard input in an environment variable visible to the context local to that thread. This environment variable is “CONTENT_LENGTH”. Therefore, gaining access to POST data in C/C++ requires a three-step process that will work in all UNIX and Microsoft development environments:

```
long   iContentLength =
atol(getenv("CONTENT_LENGTH"));
char  *szFormData = (char *) malloc(iContentLength * sizeof(char));
bzero(szFormData, iContentLength * sizeof(char));
fread(szFormData, (iContentLength - 1) * sizeof(char), 1, stdin);
```

A given browser document can contain both GET and POST data; therefore, both mechanisms may be used at the same time. In the <FORM> tag, a target may be specified which contains GET data, and the data contained in the form will be passed to the target as POST data.

How Your Program Passes Data Back to the Web Server

With the data from the web page obtained, your program can now perform all of its processing and can tell the web server what to reply. That reply may be a simple plain text message, an HTML document (the most common form), an image (typically in GIF or JPEG format) or any other complex data type. These data types are referred to as MIME types, and a standard subset is recognized worldwide on almost all browsers in use today. The mechanism by which your program passes data back to the web server (for transmission to the client browser) is by writing that data out to the thread's standard outstream, the

same mechanism that is used to write characters to the screen in your favorite language. The format for this data is simple:

```
Content-type:[SPC][MIME-type];[CR][CR][Document-Data]
```

First, your program must declare the MIME type. The most common MIME types are as follows:

- text/plain—Plain text that is output as block characters with exactly the alignment used when transmitted (no word wrap).
- text/html—Standard HTML document text.
- image/gif—Image encoded using one of the Compuserve GIF image specifications (it should be noted that the format uses Lempel-Ziv compression technologies which may not be in the public domain and may require the software producer or software user to license the software from the owner of the patent, which is Unisys).
- image/jpeg—Image encoded using the JPEG image standard.

Second, your program must send a semicolon (“;”) followed by two carriage returns (“\n”).

Third, your program must prepare the body of the document you wish to transmit. It may be the content of a plain text or HTML document or the binary data that makes up the raw data block of a GIF or JPEG image.

Therefore, getting the web server to send a simple reply can be as easy as:

```
printf("Content-type: text/html\n\n<HTML><HEAD>
   </HEAD>"
"<BODY><H3>My Quick Test Page</H3></BODY></HTML>\n");
```

That's it; those are the basics for telling the web browser what to reply to your client's request. There are, of course, some cute things one can add to this basic format that lends a measure of control over how the document is rendered. One example is the addition of the “charset=” qualifier after the MIME type (right before the carriage return), which ensures that the browser will render the HTML document being transmitted using the appropriate character set (examples are “ISO-;9660-;1”, “ISO-;9660-;2”, “KOI-;8”, “WIN-;1225”, etc.). Therefore, the savvy programmer may wish to send out the document like this:

```
           printf("Content-type: text/html;
      charset=KOI-;8\n\n"
                 "<HTML><HEAD></HEAD><BODY><H3>"
                 "<BODY><H3>Maya malinkayaproba
       </H3></BODY></HTML>\n");
```

Pushing Continual Updates to the Browser

Every so often the purpose of a web page is to monitor some long and involved process that typically takes longer than one time-out period to complete or to generate a full update. This is another situation that can be dealt with well in legacy languages like C/C++ and Fortran. The idea is to force the web server to keep the TCP pipe open to the browser and to keep pushing new documents down to the browser at an interval specified by your program.

The formula to accomplish this, given here, is specific to the Apache web server, which as we all know, is the most popular HTTP dæmon used in the Linux world to date. If you are unsure whether this will work with your particular HTTP dæmon, try it and let me know. Here are the steps:

1. Rename your program's binary to begin with the characters “nph-;”. This means that if the binary of your program is named “update.cgi”, then change its name to “nph-;update.cgi”.
2. Transmit the HTTP header that the web server normally hands to the web browser (this is done for reasons that will be explained below):

```
printf("HTTP/1.0 200 Okay\n");
```

1. Define the MIME type of the document as “multipart/x-mixed-replace”:

```
printf("Content-;Type:multipart/x-;mixed-;replace;"
        "boundary=SoMeRaNdOmTeXt\n");
```

1. Initiate the first document transmission by passing the token declared in “boundary”:

```
printf("\n—SoMeRaNdOmTeXt\n");
```

1. Send the next document update. This is simply a document that should be displayed until the subsequent transmission goes out along the same open connection at some point in the future. The update is followed by another instance of the token declared in “boundary”:

```
printf("Content-type: text/html\n\n<HTML><HEAD>
    </HEAD>"
"<BODY><H3>Update #%d</H3></BODY></HTML>\n"
"\n-SoMeRaNdOmTeXt\n", Count++);
```

1. Flush the standard out buffer:

```
fflush(stdout);
```

1. Repeat steps five and six until all updates have been transmitted. On the last update, do not transmit the token simply flush standard output and

exit. This will leave the last update in the client browser's window after your program exits.

A simple example of a program that uses server-side push to count on your browser's screen from one to ten with a delay of one second between count updates is shown in Listing 1.

Listing 1. Count to Ten

In order to explain how this works, it is necessary to understand a little bit about what the server does in the background. Up until this point, your program's output was verified for validity (i.e., a proper MIME type, proper separators, etc.) and was passed on to the client browser with some additional HTTP headers pre-pended to it. In order to take more control over the web server/client browser interaction, we must ask the web server to stop performing these validity checks and to stop adding its normal headers. This is what the “nph” stands for your program's new filename No Parsed Headers. When the name of your program begins with the letters “nph-”, this means that the web server now assumes that your program is responsible for performing all of the validation checks and header transmissions that would normally be the responsibility of the web server. The web server will simply keep the TCP pipe open to the client browser and grab data as it comes out of your program's standard output stream and pushes it down that TCP pipe to the browser. We are now in a position to understand what is happening in step two; this is a required header that is normally transmitted by the web server and was completely transparent to the program hiding behind the CGI.

Next, we must tell the client browser to expect continual updates, not just one single burst of data...and, therefore, it must not close the TCP pipe once the first document has been transmitted. This is accomplished by specifying the MIME type of the document as being “multipart/x-;mixed-;replace”. In addition, we need to tell the browser how to differentiate between the documents in the stream of multiple documents about to be transmitted. This is accomplished by attaching the qualifier “boundary=SoMeRaNdOmTeXt” to the MIME-type declaration. This tells the web browser that anytime it encounters the sequence of bytes “--SoMeRaNdOmTeXt” in its input stream, it should stop and assume that the following data will describe a new document that will replace the one which currently exists in the document window.

The string that separates the end of one document transmission and the beginning of the next is usually referred to as a boundary token, and this token is normally much more complicated than the one shown in our example here. Normally it is a 50- or 60-byte-long alphanumeric string generated by a randomizer function and will be presented later in this article. The string should

be sufficiently long and its contents sufficiently random so as to minimize the chances that it will accidentally occur as part of the body of your document.

Finally, once the document has been pushed to standard out and the boundary token has also been pushed out, it is necessary to flush the output buffer in order to ensure the data gets sent to the client browser. If this is not done, the data will not be sent until the stream's buffer implementation that your operating system uses overflows, and a flush is triggered by the operating system.

Bulletproofing and Parsing Web Server Input

The greatest deterrent to writing web programs in these legacy languages, and probably the greatest driver behind the development of Perl and PHP, has been the difficulty and security risks involved in developing applications that have the smarts and know-how to parse and avoid hacker attacks when data is passed to them from the web browser using only environment variables and the standard input stream.

The first thorny issue that must be solved is an easy and memory- efficient method of parsing up this data so that one can simply select the field they are looking for and obtain the data in a one-shot, one-kill fashion. In addition, certain security issues need to be plugged, such as data overruns from a misbehaving client browser intended to overwrite application memory with the overrun data (or deny service).

I present here, for your browsing pleasure, a series of functions that provide just such a safe and secure one-shot, one-kill approach to obtaining POST data in these legacy languages. The specific example I present is in C but can easily be ported to Fortran or wrapped for C++:

```
char  *TextField    = GetFormStringValue("TextField");
int    NumericField = GetFormIntegerValue("IntegerField");
float  FloatField   = GetFormFloatValue("FloatField");
```

The source for these functions is shown in Listing 2 and the source for their support functions is shown in Listing 3. [Due to the length of Listing 2 and 3, they are available from our ftp site, ftp.linuxjournal.com/pub/lj/listings/issue82.] All of these functions have been tested to work equally well in UNIX and Windows development environments and both compensate for both buffer overruns and underruns. When any of these functions are first called, dynamic memory allocation to capture and parse the POST data is performed in the background. Its parsed form is then held in memory and, on subsequent calls to any of these functions, simple linear scans of the fields in this memory space are performed. Memory allocation is performed only once, and all conversion of escape sequences and special characters is performed linearly

within this memory space (no other temporary space is used to accomplish this).

Since the example shown here is in simple C, which cannot provide automatic desructors the way that C++ can, it is necessary to call one cleanup function when your program exits: ReleaseFormData()

This is necessary to release the dynamically allocated memory buffer. If these functions are ported to a C++ class, it is simply necessary to call this function in the destructor method of the class to which the POST data access functionality is ported. Therefore, a simple framework for your legacy language CGI program is shown in Listing 4.

Listing 4. Legacy Language CGI Framework

Future Topics

Of course, we have only scratched the tip of iceberg with what is possible when you unleash the power of a fast and efficient language like C/C++ for development of web application, without the added drag of having to perform all of the mundane jobs normally performed by a script interpreter. It is easy for us to see why we need to expand this discussion to include the following:

- Using the local file system to maintain “state” for your CGI programs.
- Why state can be maintained on the local file system in Linux without the concerns for disk overhead one might have on other operating systems.
- Creating, modifying and destroying cookies on your client browser from your CGI programs.
- Setting up security so that only you and the CGI program can access the state information in the files on your local file system and nobody else.
- Thinking ahead to lightweight threads and fast-CGI.

Dan Teodor makes his living doing management consulting for PriceWaterhouseCoopers in Houston, Texas. He has been a closet Linux geek since his graduate school days and the Slackware 0.x kernel releases. While big on large scale web app deployment and funky commerce architectures, he dreams about skiing New Mexico and launches a new bankrupt dot-com once a year.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Porting from IRIX to Linux

George Koharchik

Brian Roberts

Issue #82, February 2001

Coding for portability to Linux: examples from the ACRT land vehicle port.

Linux is a superset of the POSIX (Portable Operating System Interface for X) specification. It runs on commodity PC hardware, MIPS systems, Suns, Macintoshes and even IBM mainframes. It's sufficiently UNIX-like that much UNIX code can be ported over with little change.

The Advanced Concepts Research Tool (ACRT) is a descendant of the ARPA Reconfigurable Simulator Initiative (ARSI). It's primarily IRIX but has NT versions for many of its modules. ACRT is a vehicle simulator that can be reconfigured into different vehicles. Raytheon supplies the land vehicle simulators, which currently are the M1A1 and M1A2 tanks, M2 armored personnel carrier, M113, M577, HMMWV and the Future Scout. Reconfigurability is done by using “Erector Set”® type hardware, soft panels for controls, serial line inputs for grips and a common set of core software.

The Erector Set hardware looks like a scaled-up version of its namesake. This allows the crew seats, controls and monitors that can be placed in a variety of positions to emulate the vehicle interiors.

We use the term soft panels to refer to touch-screen activated control panels that are used in place of the real buttons and switches in the actual vehicles. These are designed to place the controls as close as possible to the real vehicle. This minimizes the amount of hardware that is vehicle-specific (currently only grips).

Grips are either actual crew station hardware from the vehicle or copies with simplified interior wiring. These provide switches and analog outputs used to control some vehicle functions. The outputs from these are converted to

voltages in the range of zero to five volts DC and fed through a BG Systems Cereal Box. The Cereal Box acts as an analog to the digital converter and feeds values through serial lines to the host computer. Finally, the core set of software used by all vehicles makes it easier to develop a new vehicle or modify existing vehicle-specific modules.

At startup, the system runs different software modules on different hosts. These different modules include things like ground motion (how does the vehicle move), weapons models (how does a weapon behave, where does the round go) and control panels for the crew to interact with. The system is kicked-off from a single host and then starts up process managers (host_mgr) on other hosts through the UNIX **rsh** command. These host_mgr processes then parse through configuration files to see what runs on their local host.

The system uses message passing to move information between the different modules. Between modules on the same host, the messages are buffered in a memory-mapped file as a form of shared memory. Between hosts, UDP sockets are used to pass information. A gateway system translates between Defense Interactive Simulation (DIS) messages and the internal vehicle messages. This allows the simulator to interact with other simulators using the DIS protocol (see Figure 1).

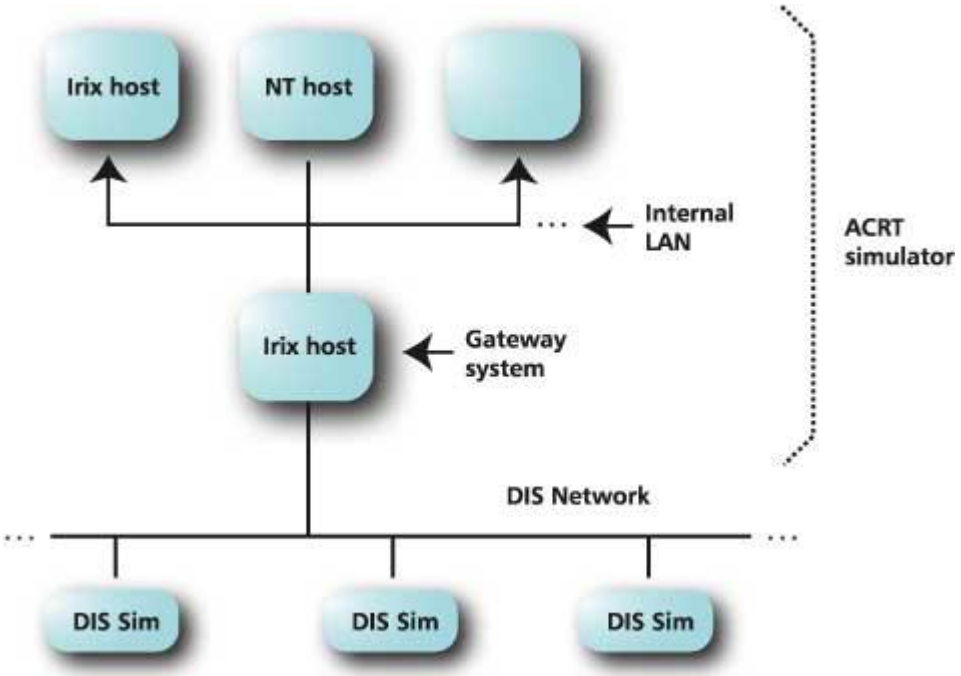


Figure 1

Objectives:

A) Demonstrate an image generator using Performer Linux (Mongoose)

What started this project was SGI's release of its Performer visualization/ simulation tool for Linux (this version is called “Mongoose”). Since our simulator's image generator (the part of the system that draws the out-the-window and sensor views) is based on Performer, the largest part of the work for a Linux port was done for us. What remained was to port the code that used Performer, the libraries and tools common to all vehicles (known as the Tiger core), and enough of the vehicle so that it can interact with its environment (to move, see other vehicles and utilize special effects like weapons fire and detonation).

B) Dual-compile whenever possible to shrink codebase

To make it easier to maintain, we decided to maximize the amount of code that would compile under either IRIX or Linux. This shrinks the codebase to a more manageable size. The theory is that each module would have a directory containing source code and under it would be subdirectories for architecture-specific items (compiled object files, libraries and executables). A Makefile in with the source files uses the OSNAME environment variable to bounce any **make** commands to the correct subdirectory depending on the architecture of the host (see Figure 2).

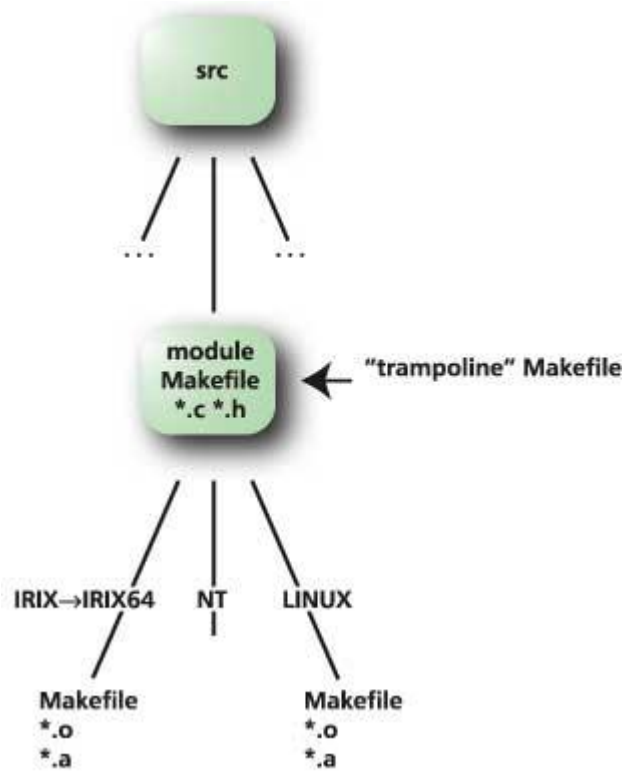


Figure 2

C) Arrange files so that machines of different architectures use the same directories

To keep the configurations as similar as possible across platforms, the files are arranged so that machines of different architectures use the same directories. For distinctions that can be made at compile time, there are different directories for compiled libraries with which modules link and where the executables are kept. Using environment variables for the BIN and LIB directories that incorporate the architecture name can help with that. For instance, on the IRIX systems, the binary directory (stored in $ARSI_BIN) might be **/apps/projects/ACRT/bin/IRIX64**. On Linux systems, it might be **/apps/projects/ACRT/bin/Linux**. Since the path is stored in the same environment variable, scripts and Makefiles can use it without regard for which type of system they are running (see Figure 3). Environment variables are also available at runtime to allow code to make decisions that can't be made at compile time. By embedding environment variables in configuration files, we can use the same configuration file for different architectures.

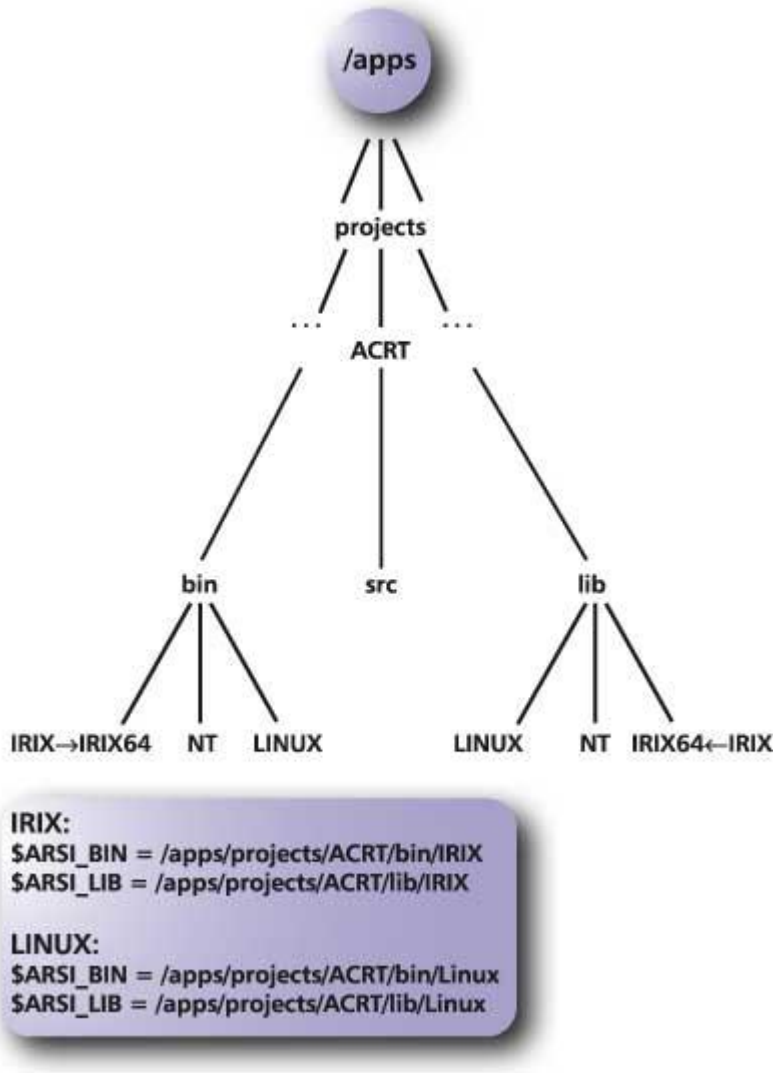


Figure 3

Portability of source files was an issue for the include files and modules that dual-compiled IRIX/NT. For IRIX to use the files, they had to be in UNIX format

(the CR/LF issue) instead of DOS format. Fortunately, MS Studio accepts files in UNIX format, so we settled on that. Adding Linux didn't change this equation since Linux handles both DOS and UNIX formats. Note, however, that shell script files must be in UNIX format.

D) Overcome incompatibilities

We first tried to use POSIX functionality. When that was not applicable, we opted for the greatest common denominator.

E) Simplify maintenance

To keep maintenance simpler, we tried to avoid convoluted #ifdef-ed regions of code, be careful about variables used and set in each branch of code and verify that the flow of control was not drastically different between branches. When possible, #ifdef-ed code was moved to a separate file to minimize the scope of its effect on variables.

Finally, different compilers are sensitive to different things. A condition that goes unremarked on one compiler may cause a warning on another. Investigate and understand your compiler's warnings. They may lead you to an error (in which case it's easier to fix at compile time than to track down at runtime). Warnings may show you where intent differed from implementation. Warnings can also hide errors. Picking out the serious message in the midst of a page of warnings about unused variables is harder than taking out the unused variables.

Software and Hardware Used

This section reflects what we did in March of 2000. See the Afterward section for more current information.

A) Red Hat 6.1

The KDE Workstation installation option provides most of the packages you need.

B) Metrolink Motif

We experimented with Lesstif. While it works well for Performer and lets us compile our other tools, it didn't handle updating some of our scrolling windows correctly. The developer's console (a tool for tracking messages) has a scrolling window that should update with message traffic even without mouse interaction. The Lesstif version didn't, but the Motif version did.

C) Mesa Graphics Library

We used the Mesa Graphics Library as an OpenGL look alike. Make sure that you follow their directions for making the widget set after you install Mesa.

D) Viper V770 Graphics Card.

We used the Diamond Viper V770 graphics card and the hardware accelerated Mesa drivers and libraries. At that time (January 2000) the GeForce 256 card was supported under Linux, but there were no hardware accelerated OpenGL drivers for it. More recently, XFree86 version 4.0 is out and while hardware-accelerated OpenGL is available for the GeForce card, setting it up can be tricky.

E) XFree86 Drivers for Hardware Acceleration

We used the XFree86 SVGA server and the XFree86-rivaGL-3.3.3.1-49riva drivers in January. OpenGL Linux drivers for the GeForce card are available from nVidia (see Resources).

F) SGI's Performer Package

Downloaded and installed per their instructions (see Resources web pages).

G) ACRT Version 1.06 (our simulator)

See Specifies for details about architecture and operation.

Tool "Equivalents"

SGIs generally offer a cushier environment than Linux. They're also working on porting more of IRIX's tools and features to Linux. For those accustomed to the SGI environment and new to Linux, here are some of the corresponding tools that we used when porting:

A) Xdiff -> Mgdiff

Do a Web search for this one. My recollection is that it requires Motif, and Lesstif won't work.

B) dbx -> gdb

Your basic command-line debugger.

C) cvd -> xxgdb

SGI Casevision debugger has a nice graphical interface to a whole suite of tools. While there is no Linux equivalent for Casevision, the xxgdb front end for dbx provides a similar debugging environment. There is also a tool called “DDD” (the data display debugger) available that we didn't use because we're already familiar with xxgdb.

D) gr_osview -> xosview

SGI's system monitoring and status tool, **gr_osview**, encompasses more than **xosview**, but xosview is similar in the ability to see processor and memory status, system load, etc.

E) editors, shells, etc.

Emacs, vi, edit, tcsh, bash and the like are all pretty much what you'd expect (I imagine there are some differences in the dark corners of the shells, but our system didn't use enough of them to run into the differences).

F) xwsh -> xterm

I can only say “equivalent” here with the proviso that the xwsh “-hold” option (which causes the window to persist even after the command running in it with “-e” has ended) is not available for xterm.

G) PCNFS -> SAMBA

The original system shared files over NFS (requiring an add-on package for NT). In this work we shared files between Linux and IRIX over NFS and used SAMBA to share with the NT systems.

H) Lint

The lint debugging tool is bundled as part of IRIX but is not bundled with Linux, though commercial versions of lint are available for Linux. There is also a package called LCLint that seemed like overkill for our work (porting legacy code). For our purposes, we used gcc and its options for better debugging and error detection:

```
gcc -ansi -Wall -Wstrict-prototypes
        -Wmissing-prototypes \
        -fsyntax_only -pedantic -O2
```

Larch C Lint is available for Linux and does have better debugging than this, if you want to go the extra mile (see Resources).

I) ICS' Builder Xcessory (BX PRO)

The C-code generated by the IRIX version 4 (and probably newer versions) of ICS's BX GUI builder compiles on Linux. Code generated by older versions needs to be tweaked. There is also a Linux version of BX available (see Resources).

J) Virtual Prototypes' VAPS

VAPS is a control panel building tool. There is no Linux version yet but VAPS does have an NT version. So my group ported IRIX VAPS panels to NT, but that is a separate story.

Specifics

A) Scripts and Shell Environment

It looks like Linux (or the tcsh shell) resets the processes limits when you “rsh” to another host. This kept me from getting core files. To get around this, I added a few lines to my .tcshrc file:

```
limit coredumpsize 1000000 kbytes <----
source ~/environment
setenv PATH ${PATH}:.
```

The environment file referenced here is a collection of environment variables for the simulator that is set at shell start up. [Due to the length of Listing 1 it could not be printed here, but is available for downloading at our FTP site: ftp.linuxjournal.com/pub/lj/listings/issue82.]

Using the results of uname, we can transparently point environment variables at the architecture-specific directories for binaries. Scripts, Makefiles and programs can then use those environment variables without each having to check the architecture type themselves. When we added Linux into the mix, we had to add some additional variables:

```
* $ARSI_SERIAL_PORT_ONE, $ARSI_SERIAL_PORT_TWO
```

The names of the serial port devices differ between IRIX and Linux. IRIX **/dev/ttyd[N]** roughly corresponds to Linux **/dev/ttyS[N-1]**,

```
* $ARSI_CTDB_FILE_SUFFIX
```

the suffix of a binary data file that is different between big-endian (SGI MIPS) and little-endian (Intel X86) systems.

B)Makefiles

Makefiles are the part of the system that got beat up the most with this port. The IRIX Makefiles are moved from the directory with the source code to the IRIX64 subdirectory, and VPATH is added to point back up to them.

VPATH (“view path”) in a Makefile is like the **-I** directives that tell the compiler where to look for include files. VPATH tells make where to look for source files —in our case, up one level. Another consequence of moving the Makefiles is that you probably have to change directories before some of the regular shell commands in Makefile (cp, lint, etc.) that operate on the source files will work.

We recommend using SGI's **smake** for the IRIX Makefiles that need to use VPATH. We'd started using **GNU Make** hoping to integrate it with NT. However MS Studio uses **Nmake** instead, so that payoff never occurred. Smake understands VPATH and the common IRIX Makefile macros. The only anomaly we can report is one that occured with some of the **clean** options. If the **make clean** option is set up like this:

```
clean:
    -rm -f *.o $(TARGET) *.a *~*
```

and **rm** can't find anything to remove, smake will sometimes (incorrectly) exit with an error about “exit badly formed number”. We don't know why it does that, but adding an explicit “exit” statement fixes the problem:

```
clean:
    -rm -f *.o $(TARGET) *.a *~* ; exit 0
```

IRIX has a handy feature that lets you specify an alternate make program to run. The name of the program is put on the first line of the Makefile following a “#!” (much like the way script files name their interpreters). So making the first line of the IRIX Makefile:

```
#! smake
```

will cause the default make program to invoke smake to process the Makefile.

Dependencies are also tracked differently between IRIX Makefiles and GNU Makefiles. The IRIX Makefiles generate a single Make.depend file that lists all the dependencies for all the source files. In the version of GNU make that ships with Red Hat 6.1, dependencies are kept in separate files for each source file. For instance, if you have a file ground_motion.c, there will be a corresponding dependency file ground_motion.d. These can be automatically generated with a .d.c rule. (See the GNU Make documentation for details and how you can combine the .d extentions into a Make.depend file.)

Compiler options are also different between the IRIX and GNU compilers. Here's the changes we made:

For IRIX: Use GNU: Purpose: --------- -------- --------fullwarn -Wall -Extra warnings, error checks-MDupdate -MD -Update dependencies-xansi -ansi -Support ANSI C

Using the **-g** option with the compiler for debugging is serious now. The IRIX debugger was still able to give you information even if the **-g** option wasn't specified. For GNU, you really need to include the **-g**. On the Linux side, -D_BSD_SOURCE may be needed if you're doing something that uses BSD functions (like **strncasecmp**).

The permitted ordering of compiler options is different. IRIX seems to like the libraries to be last, gcc doesn't seem to care.

One of the handiest Makefile macros, $$(@F), is only partially available under GNU make. Make allows you to use $(@F) to extract the file name of a target. IRIX make and smake allow you to use $$(@F) on the dependency line of the Makefile. GNU make allows this only in the action clause, but it does allow you to use the pattern-matching macros to get the same effect (compare the IRIX and Linux Makefiles in Listings 2 and 3, which can be seen at ftp.linuxjournal.com/pub/lj/listings/issue82).

There are some additional directories in Linux for libraries, and they include files that may need to be added to your Makefile:

- X libs in /usr/X11R6/lib
- X headers in /usr/X11R6/include
- OpenGL in /usr/include/GL, /usr/X11R6/GL

Make sure that CFLAGS has **-c** if you have separate compile and link steps. Otherwise GNU defaults to trying to do it in one step. If you get “storage size not known” error when compiling, try removing the **-ansi** specifier from the command line.

You can use the same Makefiles for IRIX and Linux, use xmkmf to generate architecture-specific Makefiles in both IRIX and Linux.

Let's turn our attention from Makefiles to C source code. As a general rule, the SGI compilers are more tolerant than the GNU compilers. Expect your code to have to be closer to the standard to pass the GNU compilers.

The following are items we modified from the IRIX code when porting to Linux.

1. Bstring.h, bcopy and bzero are not POSIX. We replaced them with their POSIX counterparts:

```
bstring.h -> string.h
bcopy(a, b, nbytes) -> memcpy (b, a, nbytes)
bzero(a, nbytes) -> memset(a, 0, nbytes)
```

GNU does have these in string.h instead of bstring.h, so this is not strictly necessary (though if you don't, you'll need to conditionally include “bstring.h”). Note that IRIX keeps “select” in bstring.h, while Linux puts in unistd.h.

2. Here are some things that generate warnings. Since the GNU compiler is more vocal than the IRIX, we fixed these:

- Main should return int.
- Watch for uninitialized vars.
- Parenthesize defensively.
- Format specs in printf/scanf: args not matching var types -scanf using the wrong type can get you in trouble
- sprintf(astring, "") -> astring[0] = 0;
- 2D array initializers need braces to be ansi -int a[2][2] = { {1, 2}, {3, 4} };

3. There is no “recalloc” in POSIX:

- Replace recalloc with realloc and use memset to zero out the additional memory.

4. Added NULL timezone as second argument to gettimeofday

- gettimeofday is not POSIX, but SGI, Linux and NT all accept it and timezone as the second argument.

5. Change sginap to POSIX usleep:

- sginap (hundredths) -> usleep (hundredths * 10000).

6. Linux “select” decrements timeval structure:

- The documentation for some other UNIX systems say they may modify timeval in the future: with Linux, the future is now
- Don't use the same timeval struct on successive calls to “select” without refreshing it.

7. Added __ANSI_C__ parseargs.h for Linux:

- Parseargs was an argument parsing library last maintained by Brad Appelton in 1991. It supports multiple platforms and tries to figure out what's available on each system (sort of a primitive precursor of the GNU configure utility).

8. Code involving the font manager (fmclient.h and variables of type fmfonthandle) was commented out—I'm not sure if this was an IrisGL holdover.

9. Ulocks.h is commented out. Possibly another IrisGL hold over?

10. Code involving sproc (SGI's lightweight process model) is migrated to POSIX threads (pthreads):

- Use -D_REENTRANT on Linux compile.
- Add library -Lpthread to link step.
- Use XInitThreads() if threads are used in an X application.
- Get the patches for gdb from the LinuxThreads page.
- Be aware that the LinuxThreads Library uses SIGUSR1 and SIGUSR2 for its own purposes. If your application uses these signals you might have to look at some other mechanism. In the worst case scenario, use the Linux **clone** function.

11. Serial port handling used for grips and touch screens migrates to the POSIX interface.

- See man termios for overview.

12. IRIX has a high-speed malloc library -lmalloc.

- No high-speed malloc lib on Linux (that I'm aware of) so lmalloc is dropped.

13. Fabs, fsqrtf, fmod are not in math.h but are in the library.

- Fabs is POSIX, fsqrtf is not
- We suggest simply using sqrt
- fmodf
- use -D_BSD_SOURCE to get at M_PI, etc.

14. fabsf -> fabs in Linux.

15. fsin -> sin in Linux.

16. fcos -> cos in Linux.

17. fceil -> ceil & do math in double.

18. Socket level differences in your header files:

- ioctl in IRIX unistd.h, Linux has it in sys/ioctl.h.

- SIOCGIFCONF in IRIX sys/sockio.h.

19. fcntl FNONBLK --> POSIX O_NONBLOCK for both IRIX and Linux.

20. sigsend -> kill for both IRIX and Linux

21. There is sysconf shell-command in Linux

- Our spatial database tries to obtain RAM size to know how much memory it can afford to use.
- In Linux, use the contents of /proc/meminfo instead.

22. Non-POSIX IRIX flock_t vs. Linux struct flock:

- Use “struct flock” on Linux.

23. Linux **mmap** doesn't autogrow.

- Autogrow is an optional part of standard (IRIX implements it, but Linux does not).
- Fill the file with zeros to the right size before mmap-ing.

24. prctl PR_TERMCHILD becomes PR_SET_DEATHSIG.

25. POSIX signal handling is different from the BSD signal handling we were using.

- Move to POSIX; man sigaction for details.

26. IRIX oserror Ý POSIX errno.

27. There is no sysmp command for Linux multiprocess control.

28. There are no Performer arenas in 2.3 (Mongoose)

- All performer in one process.
- Arena pointers will be NULL.
- pfMalloc and company allocate off the stack.

29. xtouchmouse (a touch-screen driver that turns touch-screen data into X mouse events) works, but can do no button presses on the KDE root window, just on regular windows.

- Unmatched button presses and releases on the root window cause the window manager to hang—tip 'o the hat to John Mellby for solving that

- We chose not to use the built-in touch-screen driver support so that we could interoperate with the IRIX systems at that level. Once, when we integrated Future Scout with FBCB2 running on an SCO PC with no compiler, we were able to send the SCO's display to a monitor and plug that monitor's touch-screen into an IRIX system. The touch-screen inputs to xtouchmouse generated X events that were sent to the SCO system allowing us to interact with it.

30. Touch-screen calibration program that figures out the mapping from touch-screen device coordinates to X coordinates is converted from IrisGL to OpenGL.

- If you still have IrisGL code, try SGI's toogl program to help with the conversion. You might have to convert to Motif if you want to suppress the window borders.

31. Replace Iris GL window management with the X Windows System. Here are some things that either the SGI compiler just let us get away with or are a function of different versions of Motif:

- In order to use glwMDrawingAreaWidgetClass (instead of glwDrawingAreaWidgetClass) you need to include a library that defines __GLX_MOTIF to make sure the Motif libraries are included or just use the glwDrawingAreaWidgetClass if no Motif functionality is required. “M” indicates Motif and has the additional functionality of Motif's XmPrimitive widget class.
- In XtPopup, you cannot use NULL for second param. Use XtGrabNone instead. This parameter specifies the way in which X events should be constrained.
- Use the X Toolkit Intrinsics method for setting up a window instead of the X Lib method in order to create a borderless window. The borderless window needs to use resources to let the window manager know that it doesn't need to manage the border. The resources are not accessible with the X Lib method.
- Remove IrisGL device.h.
- Use -lMesaGLw instead of -lGLw for library in the Makefile. This library has the OpenGL draw area widgets.
- Note that there seems to be a bug in the current (as of February 2000) alpha TNT2 drivers that keeps glColorMask from working. If you're working on this now, I'd suggest switching to the newer drivers for XFree86 version 4 and the latest drivers.

32. Finally, if you do need to compile something conditionally, there is an automatically defined symbol for #ifdef (no -D needed):

```
#ifdef __linux__
    Linux specific stuff
```

```
#else
    IRIX specific stuff
#endif /* __linux__ */
```

The bulk of the data our simulator dealt with wasn't a problem. Much of the configuration files are ASCII and need only minor tweaks (like environment variables) to be simultaneously usable on IRIX, Linux and NT. In the case of Mongoose, the loaders did the byte swapping for me, so it didn't matter if I was on a big-endian SGI or a little-endian Intel box—the same visual models worked.

There were a few cases where endianness became an issue. We use the Compact Terrain Database (CTDB) files produced by the US Army Topographic Engineering Center. The database files we used were binary. Fortunately they came in big and little-endian versions with different suffixes. We could tell the system which to use by storing that suffix in an environment variable that's evaluated at runtime. The S1000 libraries are not implemented in Linux, just because they are larger and more complex than time allowed. Our Spatial Database has a binary representation of the shapes of different vehicles, so that needed to be converted. Finally, we use the BG Systems Cereal boxes to read our crewstation controls (grips). The Cereal box reads analog voltages from the crewstation controls and transmits its data over a serial line to the host computer. Since the data was assembled byte-wise into floats a little bit of their library needed to be modified. BG Systems was very helpful in that effort.

Once we had everything compiled, there were some surprises at runtime. A runtime error from the GNU compiler helped us find a case where we were accessing memory after it had been freed. IRIX will sometimes let you access memory even after you have freed it (see the IRIX mallopt function for different settings of their memory allocator). There is a **mallopt** function in Linux, though the man pages don't list it. You'll have to use the GNU “info” pages for that. Another feature of the GNU C library useful for tracking memory errors is **mcheck.** It's also detailed in the info pages.

The second thing is that the serial port device files have different names between IRIX and Linux:

IRIX: Linux: ---- ------/dev/ttyd1 /dev/ttyS0 serial port one/dev/ttyd2 /dev/ttyS1 serial port two... ...

The grip calibration programs decide based on architecture, and look for the value of $ARSI_GRIP_CALIBRATE_PORT as an optional override.

Was a case where the we had a different results from local variables in a function. We had a string next to an array of longs. The code contained an error where the string was overflowed. On the IRIX systems, the code worked fine.

On the Linux systems, it did not. On further investigation, we found that the string was terminating against bytes in the array of longs. In the bigendian systems, those bytes happened to be zeros (making the system appear to work properly), while on the little-endian systems, they were nonzero, causing the string to appear to be corrupted. (Jim King gets the credit for finding that one.)

Finally, there is an error in the Flight loader that comes with Mongoose 2.3. MultiGen-Paradigm says that will be fixed in the next release. (A tip 'o the hat to Karen Davidson for tracking that one down.)

Status

Right now you can drive, shoot the weapons and interact with DIS entities in Paradigm's Atlantis database (credit goes to John Powers and Lori Shearer for helping with the CTDB files). The system can talk to VAPS panels running on another host under Windows NT (credit goes to Kay Chao, Kathy Jones and Vince Golubic).

Afterward

Much has changed since this work was done (first quarter 2000). ACRT can now use the newer High Level Architecture (HLA), a successor to DIS. SGI is continuing to make contributions to the world of Linux, including releasing the reference implementation of OpenGL. A new version of Mongoose is out, and we're experimenting with it now. By the time you read this, version 2.4 should be out, unifying IRIX and Linux distributions. nVidia has come out with new graphics cards and Linux drivers. The long-awaited XFree86 version 4 is out with direct rendering of OpenGL. (Prior versions took a performance hit when they had to first render to X and then to the screen.) Motif is now free on Linux systems.

A Linux version of the VAPS tool we used for soft panels is due out in first quarter of 2001.

Resources

George Koharchik mourns the passing of lisp machines and works at Raytheon. He can be reached at g-koharchik@raytheon.com.

Brian Roberts joined Raytheon Systems Company in 1995 as a full-time member of the Visualization and Simulation Technologies team. His main responsibilities consisted of design and development of new software for a reconfigurable visual simulation. He now works at Texas Instruments.

Archive Index Issue Table of Contents

Advanced search

Expanding Options for Clustering

Ken Dove

Issue #82, February 2001

The role of Linux in the future of clustering.

Ten years ago, clustering meant one thing—grouping a small number of large servers together with shared disks to host on-line transaction processing applications with better availability and, in some cases, scalability. Since then, we've seen the rise of internet-based applications and thin commodity servers, and a whole new model of open, scalable, distributed computing built upon Linux.

Many vendors in the Linux space are working on reproducing the kind of clustering that existed ten years ago on traditional UNIX OLTP systems. This may not be the right strategy. Rather, clustering should look toward the types of applications and architectures that are being deployed in the internet data center and on sets of thin Linux servers. In the internet data center—whether in a hosting environment or in a corporation—application architectures have several characteristics that are fundamentally different from the architectures of the old glass house. In particular, the internet data center architectures share some important common themes:

- Architectures are multitiered (usually web servers, application servers, and database servers).
- Each tier may have literally dozens of servers.
- Each tier has unique requirements for shared and replicated data.

In a large e-business or web content site, managing the entire distributed set of resources that provide a service over the Web is a bigger challenge than simply providing failover for individual servers.

While some applications, like OLTP or some kinds of e-commerce failover, still require shared disk solutions, modern applications, in many cases, can take advantage of software-based replication provided by middleware. What they

lack is the infrastructure to bind those solutions together and integrate them in the larger computing environment. Application management—including failover—remains one of the key problems facing today's operations manager, but the options are, fortunately, getting better.

The applications that do require shared access to data now often involve dozens of machines, potentially in multiple Points of Presence (POPs). This means that even the shared disk solutions of traditional clustering, which are based on physically shared SCSI connections, may not be adequate. Those solutions are bound by physical cabling and SCSI address limitations and usually can only service two or four systems.

The Web Server Tier

The web server tier, a stronghold of Linux, is characterized by either read-only or dynamically created data. This data can be, and often is, shared across multiple servers by using network-attached storage appliances running NFS. Each server mounts the NFS appliance and has access to the right data. For this tier, complicated shared SCSI cabling schemes are totally inappropriate—a TCP/IP network provides all the interconnect between servers and storage. However, clustering software that can monitor the health and performance of web servers, and take appropriate actions based on the input from those monitors, is still very important.

The Application Tier

In the traditional corporate glass house, applications were typically client/server and were based on a relational database. For the most part, data was not stored in the application layer—everything was put in the database. The mix is much richer in the Internet/Linux world. The emergence of Java and Enterprise JavaBeans has led developers to serialize application-layer objects in the file system, outside the database. In addition, applications now manipulate a much richer universe of data—including digital streaming media, text and other data types that are not naturally stored in SQL back-ends.

For this reason, the applications layer has shared data and failover requirements that will be met by neither network-attached storager or shared SCSI. Network-attached storage typically does not work because NFS protocols do not provide enough consistency guarantees to be suitable where multiple servers may be writing the same data. Shared SCSI is not sufficient because replication nor sharing may need to happen across many systems or across multiple POPs. In many cases, therefore, applications provide their own replication. For example, consider a foreign-exchange trading application in use at several dozen large banks. Replication of real-time trading data is maintained at the application level, combined with cluster-based failover.

The Database Layer

The database layer is the domain of traditional clustering solutions from the big UNIX vendors—Sun, HP and IBM. Originally, two servers would be connected to some SCSI disks, with one server being an active master and the other being a passive standby. In more modern implementations, multiple servers share a fiber channel or SCSI storage system and use a parallel database like Oracle Parallel Server (OPS) to manage concurrent access by each node in the cluster.

However, high-end fiber channel SANs are expensive, as is software like Oracle Parallel Server. Often, the price/performance of these types of solutions are not appropriate for the thin server Linux market or for the internet data center. The active/passive SCSI approach is also wasteful of resources and suffers limited scalability. Fortunately, there are other approaches more suited to the kinds of applications being run on Linux.

For example, database vendors have put much effort into shared-nothing software replication approaches (Informix Replicator, DB/2 Replication, Oracle Multi-master and hot standby). Using these tools, it is possible for multiple servers to participate at the database tier without having shared storage. However, the databases themselves often lack the facilities to detect failure, sequence the actions to initiate failover and guarantee application integrity. For this reason, they are best paired with a clustering solution that provides these services.

A real-life example is a chip fabrication facility running a manufacturing automation application on Intel-based thin servers. Because of the value of this application, the chip manufacturer requires three-way replication, so that even if the primary fails, there are two machines ready to take over. The solution chosen was to use Informix database-replication services running on physically separated machines without shared storage in conjunction with a cluster management engine.

The Future of Shared Storage

Finally, some applications do require physically shared disks. However, because shared SCSI remains an unusual configuration, putting together complete systems using this approach requires care—often complete hardware/software system certification by a vendor. (When was the last time you checked your disk firmware level?) Even when this is all arduously put together, as we have seen, there are severe limitations on both the number of nodes that can share data over SCSI and possible physical cable layouts. Fiber channel relaxes some of these limitations but comes with drawbacks of its own, including steep price points, problems with multivendor interoperability and a set of management issues unfamiliar to many users of Linux thin servers.

The next year will see a huge change in the face of shared storage. Industry leaders are pushing two new major storage interconnects that will make shared storage available at a price point appropriate for thin servers that will also support sharing by large node counts seen in Internet data centers. Cisco and hot startups like 3ware are backing SCSI-over-IP, which will allow SCSI block protocols to run over a switched Ethernet fabric. Intel and the server vendors are lining up behind a new I/O standard called Infiniband. It will provide a switched I/O fabric that could eventually be implemented in the chip sets included on every commodity server motherboard. Indeed, these developments are complementary—Gigabit Ethernet cards will be able to sit on the Infiniband fabric and run the SCSI-over-IP protocols. As a consequence, we can look forward to having dozens or even hundreds of systems share storage over modern interconnects in the near future. Clustering solutions that provide the basis for using these interconnects to create truly manageable and scalable distributed solutions out of sets of Linux boxes will set the tone for the data center architectures of the future.

Ken Dove is chief architect of PolyServe, Inc. Previously, he was a distinguished engineer at IBM and before that principal software architect at Sequent Computer Systems for 12 years.

Archive Index Issue Table of Contents

Advanced search

That's Vimprovement! A Better vi

Steve Oualline

Issue #82, February 2001

Be a better editor—try Vim.

The Linux system comes with a vi clone called Vim. However, this editor can do more than just mimic vi. It has literally hundreds of additional functions, including commands for a help system, multiple windows, syntax coloring, program compilation and error corrections, advanced searching and many more.

Getting Started

By default, Vim starts in vi-compatibility mode. This means that many of the advanced features are turned off. To turn on these Vim improvements you need to create a $HOME/.vimrc file.

Listing 1 contains a sample .vimrc file. You can also create one by copying your .exrc file if you have one. A zero length file works as well.

Listing 1. Sample .vimrc

There are two flavors of the Vim editor. The command **vim** starts the console version of the editor. In this version, you edit inside the terminal window in which you executed the command. The **gvim** command creates its own window. This second command is preferred because you get features, such as a command menu and toolbar, not found on the console version.

Help

One of the most useful innovations that Vim has is the on-line, integrated help.

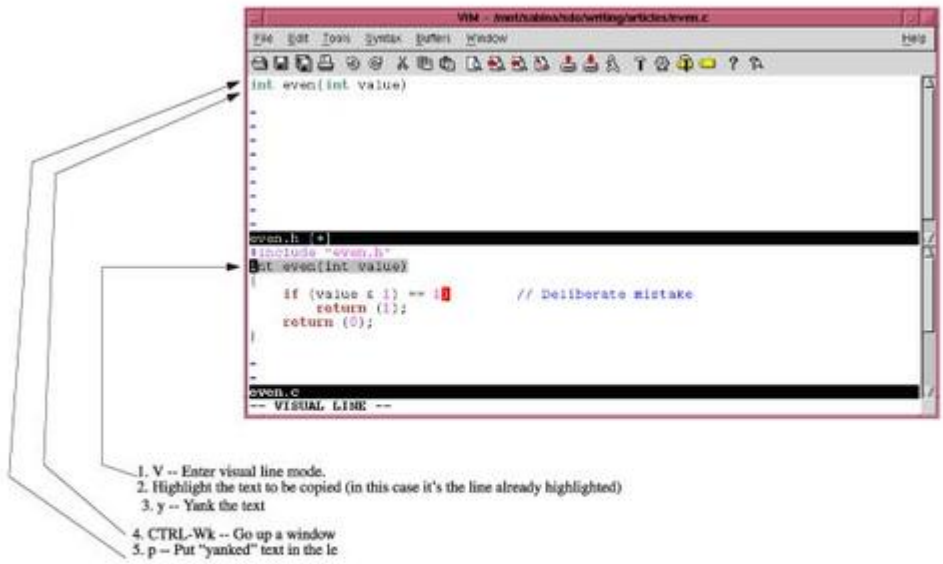


Figure 1. Vim Help Window

The **:help** command displays a help window. For specific help **:help /** displays the help text for the search (/) command (see Figure 1).

You can move through this text using the normal editing commands, such as h, j, k, l, <Page-Up>, <Page-Down>, etc.

As you scroll through the text, you'll see some lines that look like

```
/<CR> Search forward for the [count]'th latest used
  pattern |last-pattern| with latest used |{offset}|.
```

The text enclosed in vertical bars (|) is a Vim hyperlink. Position the cursor over one of these items (say |last-pattern|) and press CTRL-]. This will make the screen jump to the given subject.

To go back to where you were before the jump, use the command CTRL-T.

To get out of the help system, use the normal Vim exit commands :q or ZZ.

Creating a File

To see how some of the new Vim commands can help you, start by creating a simple program file containing the following text:

```
#include "even.h"
int even(int value)
```

```
{
  if (value & 1) == 1)       // Deliberate mistake
      return (1);
  return (0);
}
```

The first thing to notice is the changing colors of the text. This is called syntax highlighting. Each component of a program (keyword, string, constant, preprocessor directive, etc.) gets a different color. In our initialization file, this was enabled by the **:syntax on** command. If you do a **:syntax off**, the highlighting disappears.

The next thing to note as you type in the file is that you don't have to do any indenting. All the lines are automatically indented for you. This is because of the **cindent** option being turned on by the magic lines:

```
:autocmd FileType c,cpp     :set cindent
```

This applies to C and C++ files only. (Unfortunately a full discussion of autocommands is beyond the scope of this article. You can do a **:help autocmd** for full instructions.)

Note: The actual commands in the sample .vimrc file are a little more fancy than the ones presented in this section, but they do the same thing.

Undo/Redo

Now, let's have a little fun. Position the cursor on the “r” of one of the return statements and type xxxxxx. The return disappears. Now type “u” to undo the last change. The “n” returns.

The old vi editor had only one level of undo. Vim has many. Type “u” again, and notice that you've got “rn” back. Type “u” four more times and the whole word returns.

So how do you undo an undo? Through the new “redo” command:**' CTRL-r**. By typing this a couple of times, you redo the delete, which causes parts of “return” to disappear.

Multiple Windows

Now that you have a C file, you should create a header. It would be nice to be able to copy and paste the prototype from the C file into the header. With vi you couldn't do this. With Vim, it's easy.

First, bring up both files in the editor. The command **:split even.h** splits the current window in two. The top-half gets the file even.h, and the bottom even.c (see Figure 2).

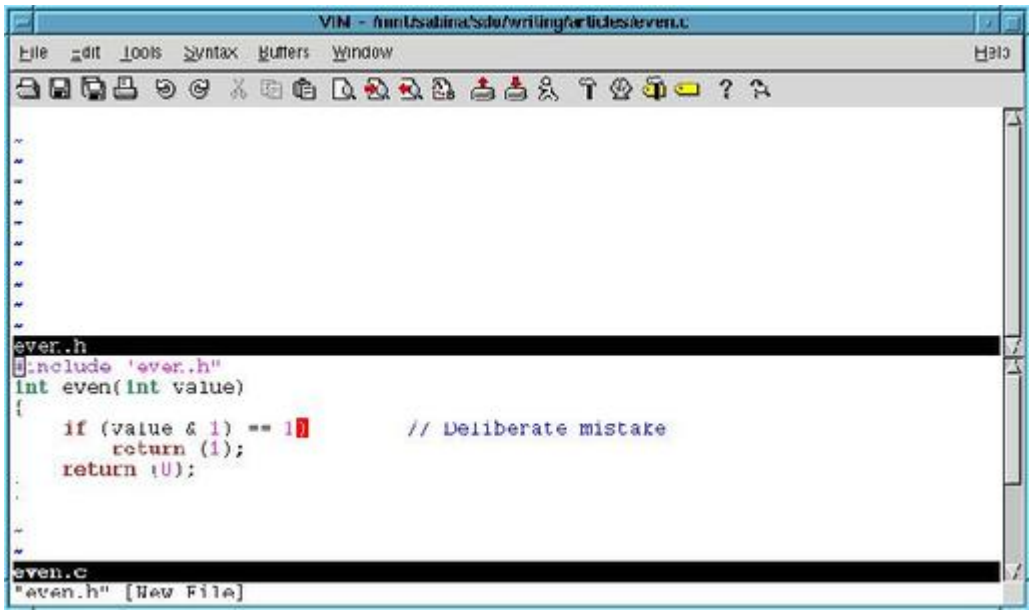


Figure 2. Multiple Windows

To move the cursor from the top window to the bottom window use the command **CTRL-Wj**. To move up a window, use the command **CTRL-Wk**.

To close a window, use the **ZZ** or **:quit** command.

Copy and Paste Using Visual Mode

Vim has one new major mode: visual mode. In visual mode, highlight the text you want to edit, then type an editing command. In this example, you need to copy the first line of the function **even** from the bottom window and paste it in the top window.

Start by sending the cursor to the bottom window (**CTRL-Wj**) and positioning it on the first line of the function **even**. Next, enter line visual mode by typing the **V** command. The current line is highlighted, which indicates that this line will be affected by any editing command you type. At this point you can use the cursor commands to move the cursor down a few lines. As you move the cursor, the text that you pass over is highlighted. Move the cursor up, and it's unhighlighted. Go past the starting point, and the text above will be highlighted.

Highlighting is done on a line-by-line basis. That's because you've entered visual line mode. If you had typed **v**, for character visual mode, you could highlight text on a character by character basis. Finally, there is block visual mode (**CTRL-V**) that highlights a rectangle on the screen. This is good if you are working with text tables because you can manipulate columns of figures on the screen.

Now, to copy a single line from the bottom window to the top one, use line visual mode (V) and highlight that line. Then use the yank **y** command to copy it into the unnamed register. Next go to the top window and do a **p** to put (paste)

the text into the top window. All you need to do now is add a ";" to the end of the line and you have an even header. Figure 3 shows these commands in action.

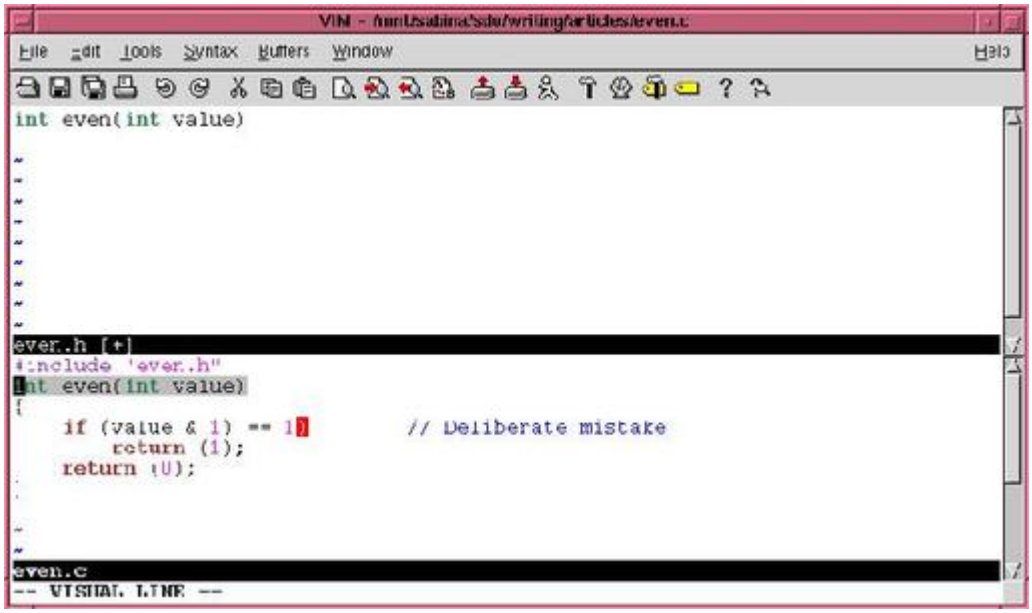


Figure 3. Copying a Line

Compiling the Program

In order to compile our program we need a Makefile. (This exercise is left to the reader.)

Once our Makefile is in place we use Vim's integration with make to compile the program. All we need to do is issue the **:make** command from within Vim and the editor will run make and capture the output.

Our program contains an error:

```
even.c:4: parse error before `=='
```

Vim captures the output of the **:make** command and parses it. From this it can see that we have an error on line 4 of even.c, so it moves the cursor to this line and displays the error message at the bottom of the screen. (It will even switch files if it has to in order to locate the error.)

This makes it easy to fix the error. To see the next error, issue the **:cn** command. If you want to see the current error again, use the **:cc** command. The previous **:cp**. Finally, if you're done editing this file and want to see the first error in the next file use the **:cnf** command.

Vim is smart regarding the location of errors. When you add or delete lines at the start of the file, Vim adjusts the location of errors at the end of the file to

reflect these changes. This avoids the problem you have with conventional editors like vi, where fixing an error at the start of a file throws off the line numbers for the rest of the file.

Finding out Where a Function Is Used

The grep command is great for finding out where a function is used and defined. Simply enter the command:

```
$ grep -n even *.c
```

and you get a printout of each line that contains the word even.

Vim has a **:grep** command that does the same thing. Actually this command is very similar to the **:make** command. It runs grep, captures the output and lets you navigate through the files using **:cn**, **:cp**, **:cc**, and **:cnf.**--just like **:make**.

Fixing Indentation

If you turn on the cindent option, Vim will indent your program correctly when you insert new text. But what about the text that's already there? That's where the **= command** comes in.

It will run a block of text through Vim's internal indentation program. (Actually, you can select which program is to be used for indentation through the equalprog option.)

Let's see how this can work to indent a basic C statement block. There are two ways of invoking the **= command**. The first is to use **={motion}**. The second is to enter visual mode, select a block of text and then press **=**.

So, one way to indent a block of code is to go to the first "{" of the block. Now enter **=%**. The **=** tells Vim to indent the text from here to where the next command takes the cursor. The next command in this case is **%**, which tells Vim to go to the matching "}".

If you wanted to do things using visual mode, you would position the cursor on the first "{", then enter line visual mode with the **V** command. Next position the cursor on the corresponding "}" using any set of commands that get you there. Finally, the highlighted block is indented with the **= command**.

Searching

Vim has some interesting searching options. The first is the incremental searching option, which is enabled with the command:

```
:set incsearch
```

Normally when you do a search, say for include, you would enter the entire command: **/include<CR>** and Vim would start searching for the string. When incremental searching is enabled, Vim starts searching when you type the first character of the search. In other words, when you type **/i**, Vim searches for the first “i” in the text. Type the next character “n”, for /in and Vim looks for “in”. As you type each character the search moves on (if needed) to find the string that you've typed so far.

The other major innovation in searching is the highlight search (hlsearch) option. If this is enabled, then when Vim finds a string that matches a search, it highlights it. Not only is the current match highlighted, but all matches are highlighted.

This is useful if you're searching for a misspelled variable name or word, because all occurrences of the incorrect word are highlighted.

If you want highlighting to temporarily disappear, you can issue the the **:nohl** command. This clears the highlighting until the next search command is entered.

Vim also maintains a search history. Let's say you've done a number of searches. Now press / to start a search, and then the <UP> key. The previous search is displayed. Press <UP> again, and you get the next oldest search. Thus using the <UP> and <DOWN> keys you can scroll through a set of your recent searches.

Keyboard Macros

One of the most powerful commands in Vim is the . (dot) command. It repeats your last edit. But this command is limited. It will only repeat a single command.

But what if you have something more complex to do? That's where Vim's keyboard macros come in. They allow you to record a series of commands in a register and then execute them.

To see how this works, let's suppose you have the following lines that you want to edit:

```
stdio.h
time.h
unistd.h
stdlib.h
```

You want to make each of these into a #include line, e.g., **#include <stdio.h>**

You start by entering the command **qa**. This causes all subsequent commands to be recorded in register a. (Any register from a to z can be used.)

Next do the edits. Go to the beginning of the line (^) insert **#include <**, then finish by adding a **>** to the end of the line. Finally go down a line to be ready for the next edit.

All these commands are now recorded in the a register. The **q** command tells Vim to stop recording. The text now looks like:

```
#include <stdio.h>
time.h
unistd.h
stdlib.h
```

with the cursor positioned on the second line. To execute the macro, use the command **@a**. The results are:

```
#include <stdio.h>
#include <time.h>
unistd.h
stdlib.h
```

The **@** command takes a repeat, so you can finish off our work using the **2@a** command.

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
```

Tags—A Program's Table of Contents

The command **ctags** (which comes with Vim) can be used to generate a program table of contents for a set of C files. (This is now a tags file.)

A typical **ctags** command is:

```
ctags *.c
```

This creates a tags file containing the location of all the functions in the C files in the current directory.

Once generated the tags file can be of great use to people editing with Vim.

Let's suppose you are editing a file and are looking at the **read_paragraph** function. As you go through the code you discover that **read_paragraph** calls **read_sentence**. You'd like to see what this function does.

If you position the cursor on the function name, and press CTRL-], Vim jumps to the definition of that function. You quickly discover that **read_sentence** calls

read_word, so you position the cursor over **read_word** and press CTRL-]. Again the editor jumps to the definition of that function.

Vim keeps track of where you've been through the use of tag stack. To see where you are in this list, use the **:tags** command.

```
:tags
# TO tag        FROM line  in file/text
1  1 read_sentence       3  read_sentence();
2  1 read_word           8  read_word();
>
```

In this example, we've gone from **read_word** (not listed), to **read_sentence** and then **read_word**. To go back, use the CTRL-T command. It pops you up one level in the tag stack.

Advanced Tagging

When you jump to a tag using the CTRL-] command, the entire screen jumps. If you use the command CTRL-W CTRL-], the current window is split and you do a tag jump in the new window.

Multiple Tags

Suppose you have five or six small programs in a directory, and you want to jump to the definition of one of the **main** functions. If you do a:

```
:tag main
```

command, the editor will jump to the first definition of **main** that it can find. This may or may not be the one you want.

The **:tselect** command displays all the tags that match a given name and lets you select the one you want. For example:

```
:tselect main
# pri kind tag               file
> 1 F C f    main              a_test.cpp
    int main( int argc, char* argv[] )
2 F   f    main              acp.cpp
    int main( int argc, char* argv[] )
3 F   f    main              add.cpp
    int main(int argc, char* argv[])
Enter nr of choice (<CR> to abort): 3
```

What if you only know part of a name? In other words, you want to go to the function whose name is “something- process-something-data”. The **:tags** command can take a regular expression. In this case the command:

```
:tag /process.*data
```

goes to the first tag matching this expression. If several functions match the expression, you can use **:tselect** to go to the right one.

Word Wrapping

Programs contain code and comments. Code has its own structure. You do not want an editor that line wraps when you write code. But comments are merely text and it's okay to wrap text. In fact in most cases, it is highly desirable to have an editor that wraps long text lines.

The Vim editor understands the difference between commands and code. It can be configured to wrap comments and leave the code alone. This is done by putting the following lines in our .vimrc file:

```
:autocmd FileType c,cpp
\     set formatoptions=croql cindent
\     comments=sr:/*,mb:*,ex:*/,://
```

(Note: continuation lines in *begin* with \.)

The **:autocmd** command tells Vim that these commands are to be executed whenever it determines that it is working on a C or C++ file. (File names ends in .c or .cpp.)

The **formatoptions** tells Vim that we want to automatically wrap comments but don't want code wrapped. The next line:

```
set comments=sr:/*,mb:*,ex:*/,://
```

tells Vim what a comment looks like. In this case we've defined the standard C comment (/*, */) and the C++ comment (//). We've also told Vim that if we are in the middle of a C-style comment, to please begin each line with a "*".

With these options set, when you type /*<RETURN>, Vim automatically puts in a " *" to begin the next line. Given the limited space allowed in this article, all I can really tell you is that this is a really neat feature, and you should play around with it. If you need more details, you can check out the on-line help text.

Conclusion

Vim is a very versatile editor. In this article we've touched on a few of the more interesting features of this editor. But we only scratched the surface. There are still hundreds of commands and options that we've not discussed. You can find out about these by reading the help text and other documentation that comes with the editor.

By introducing you to some of the new features in Vim we've given you a start toward more effective editing. How far you go from here is up to you.

Sidebar

Steve Oualline is the author of Practical C Programming, *Practical C++ Programming* and *Vim (Vi Improved)*. He wrote his first program when he was 11 years old and has been working in the software industry since then. He currently resides in San Diego where he spends his days working at Nokia Mobile Phones and his weekends working as a real engineer on a small tourist railroad in Poway, California.

Archive Index Issue Table of Contents

Advanced search

Open Source in Electronic Design Automation

Michael Baxter

Issue #82, February 2001

An interview with Stephen Williams, the creator of the Icarus Verilog compiler.

The world of open-source software is making inroads into areas beyond operating systems, Internet and desktop applications, GUIs and scripting languages. One less well-known area of open-source development is Electronic Design Automation (EDA) for Hardware Description Languages (HDLs). There are two main HDLs in use, VHDL and Verilog. Verilog is widely used for logic design and simulation in the semiconductor industry and elsewhere.

HDLs are languages for representing hardware, typically for simulation or synthesis. Representing hardware can be done at more than one level of abstraction, depending on the desired application. Looking at abstractions will help illustrate why HDLs are different from conventional programming languages like C, C++ or Java.

For the purposes of “what-if” simulation, hardware is modeled in the HDL by describing in HDL code what it *does*, e.g., how it be behaves electronically, not what it *is*, as a circuit. This kind of HDL code looks like a rather conventional computer program. A mid-level abstraction often used in HDLs is Register Transfer Level (RTL). This kind of code reflects a structural implementation at the level of registers (implemented with flip-flops or other bistables called latches) that have various logics inserted between them. The code describing the logic can be either behavioral or more like what would be used in an actual

implementation. RTL is used first to sketch an outline of the system, then to mold it progressively into a detailed implementation. Through this refinement process, the code somewhat resembles macros, and designers may even take advantage of libraries of RTL logic. Logic circuits can also be modeled at a very low level, where the implementation of the circuit is literally described in code. This is known as structural design, and it can look like long listings of assembly language.

So, with only one language, all of these levels of abstraction can be described equally well, and mixes of them are often used in design. There's an even bigger difference between HDLs and programming languages: time. In a way, HDLs describe the ultimate concept of a “program counter” because they all model time so that the behavior of logic circuits can be properly ordered, for instance as clocks progress in a system. This leads to a very large difference in language semantics. HDLs are concurrent and effectively model parallel-system behaviors. For performance, HDL compilers are often implemented as C or C++ programs. However, the compiler implements semantics for a language that inherently describes parallelism because this resembles hardware more closely.

As a result, EDA tools that use Verilog tend to have some unique qualities and requirements when contrasted with traditional software tools. To put this story in context, we offer an interview with a leading open-source EDA developer, Stephen Williams, who has written the Icarus Verilog compiler under the GPL.

LJ: What is the Icarus Verilog compiler, and how does it work?

SW: Icarus Verilog is a compiler for the IEEE Standard 1364 Hardware Description Language Verilog(r). Those familiar with the the EDA industry recognize it as similar in concept to VCS, in that it is a compiler for a language that is commonly interpreted.

It works by parsing the syntax of Verilog into an annotated parse tree that is then “elaborated” into a design graph. The elaboration process instantiates modules, evaluates and propagates symbolic constants, checks the connectivity of all the devices and produces a checked, consistent design.

The design graph is transformed by various optimizers and logic synthesis steps into a new design graph that is more appropriate for the selected target, which is then scanned by the final code generator.

Finally, the code generator scans the design and generates the desired format output. For simulation, it generates C++ that uses a simulation class library included with Icarus Verilog. It also includes an XNF code generator for sending

synthesized designs to further FPGA tools. I'm currently working on a loadable target code generator to support a variety of other target formats.

LJ: How long have you been developing Icarus Verilog?

SW: My logs show that it was introduced to CVS in November 1998. I had a few false starts for at least two years before then. If memory serves (and it rarely does), I think I was on the current path for close to a year before it got into CVS.

LJ: What was your motivation for developing this tool?

SW: The glib answer is that I have a Linux/Alpha system. That doesn't quite say it all, though. It is pretty obvious that Linux/Alpha doesn't get much respect from the EDA vendors, but it is also true that even Linux/Intel gets little attention. I just can't fathom why so many EDA vendors are so in love with Microsoft products.

My real reason is not so altruistic, though. Basically, I'm doing it because I can, and I can do it well. My set of skills seems well suited to this sort of project and, by now, even more suited. I'm coming through this with a much deeper understanding of the chip and FPGA design process than any software engineer should be allowed to have.

LJ: Why did you elect an open-source development mechanism?

SW: Access to my own intellectual property. I have done fairly large software systems in the past that I lost access to due to job changes and employee agreements. Although my current employer makes no claims to Icarus Verilog (it is understood that it is my own work on my own time) the copyright notices make that explicit, and this seems like a safer habit for everyone concerned.

Once the employer's potential proprietary interests are out of the way, I could have still done it as a closed-source personal project, but where's the fun in that?

Oh, by the way, having the source out there does seem to improve the quality of bug reports I see. I even get patches, sometimes including new ports or significant new features.

LJ: Who are some of the key people helping you with this effort?

SW:The README in the source distribution names names, but Steve Wilson has probably helped the most by managing the regression test suite. I've been told

many times that the test suite is the most important single asset a compiler writer can have, and that is turning out to be pretty much the case.

I also deeply appreciate the bug reports I receive from users. They have been high-quality, detailed and almost always accurate. It is rare that I find that Icarus Verilog is right and the user wrong, and, when that does happen, it's pretty darn subtle.

I also use bug reports and change requests to guide my priorities when I'm trying to decide what to do next. They are also a great source of regression tests.

LJ: What are some of the typical uses of Icarus?

SW: Well, it's hard for me to know because I don't have a marketing staff finding this sort of thing out. My only source of information in that regard is feedback from those who choose to contact me—mostly, bug reports.

It seems that the users who work at large institutions may have limited access to the hugely expensive HDL tools, and they use Icarus Verilog to work on library parts and subsystems at home or on their desktop Linux machines.

I've heard from some people who have abandoned their commercial tools for simulation of small to mid-sized designs for a variety of reasons, ranging from pure cost benefit to pure activism.

Smaller scale HDL users might choose Icarus Verilog because it is good enough for what they need, and the price is hard to beat. It opens HDL design to those who would otherwise be locked out.

LJ: How might Icarus compare with a commercial EDA tool?

SW: Icarus Verilog is no real threat to the higher-quality big-name tools. They've got a bit of a head start on me. It was not long ago that I thought Icarus Verilog was not competitive at all, but I find that there are plenty of commercial tools that are even more unworthy of sale.

The differences come in the language coverage, simulation performance and synthesis quality. Icarus Verilog actually stands up pretty well with language coverage and is improving still. It seems to be about average.

Simulation performance is hurt severely by the performance of the g++ compiler that compiles the generated simulation. I'm afraid it was a mistake to generate C++ as an intermediate form, so over time I must replace it with

straight C or an interpreted back end. Once the design is compiled, though, it runs reasonably well.

Synthesis in Icarus Verilog is not yet commercial quality, although some people are using it effectively. If you stick within the limits of the Icarus Verilog synthesizer, you can do nontrivial Xilinx designs. I know that some people for example have replaced Abel with Icarus Verilog in their design flow.

LJ: Are there aspects to proprietary EDA tools that make Icarus harder to do?

SW: For one thing, Icarus Verilog doesn't do everything that an EDA user needs. Sure it compiles to XNF, but you still need vendor-specific tools to map to and optimize for the part you are using. It has been hard for me to get needed interface information from the “layer below” vendors, leaving me to reverse-engineer Netlist formats. That is very irritating.

It also doesn't help any that the “layer below” tools commonly don't exist for Linux. This too is very irritating.

LJ: Does the open-source nature of Icarus Verilog provide some benefits over proprietary EDA tools?

SW: The most obvious benefit of Icarus Verilog over proprietary tools is the flexibility of your work environment. If you do a design that you know works with Icarus Verilog, you can be confident that you can buy a new computer, whatever is nifty this week, and expect to be able to use it.

I've noticed that EDA vendors are advertising licenses that don't expire, but the computer and OS you are running it on most certainly will. And, of course, Vendor X does not support Product Y Version 1 on the new system you just bought. So when you buy a new computer, you wind up buying software updates as well, and that is not only expensive but potentially risky if you are talking about, for example, a 95% full XC4013XL design.

(I've seen later versions of tools break existing designs.)

LJ: The commercial EDA industry seems slow to adopt Linux, although some have. However, EDA vendors seem to be especially reluctant to adopt or exploit open source. Can you speculate on why?

SW: Well, I imagine it might be tough to embrace open source if you have a $100,000 software product, but other than that I have no clue. All my coworkers at my day job complain constantly about being stuck with Microsoft operating

systems, but there is no choice. The FPGA vendors are completely unresponsive to this particular gripe. I wonder if I can do an open-source place-and-router?

LJ: How much code is involved in Icarus?

SW: It's on the order of 50,000 lines of C++ with some C, lex and yacc thrown in. It seems to have stabilized at about that size for the last several months to a year. In other words, I'm removing as much code as I add.

There is also a test suite of Verilog code that is on the order of 300 small Verilog tests (16,000 lines of Verilog) along with a bit of Perl to drive it.

LJ: Can you suggest some improvements to open-source development tools that would be beneficial for projects like Icarus?

SW: Well, it would be nice if g++ compiled about ten times faster. I shouldn't complain too loudly though, as MSVC++ can't even compile my compiler. I've also run into symbol table limitations with Linux/Alpha and Cygwin32 binutils.

LJ: Linux is obviously a central platform for Icarus. How hard has it been to port Icarus to other platforms?

SW: I have precompiled binaries of the last stable release contributed by porters to Solaris, NetBSD and Mac OS. Recently, a Windows port was managed using the Net release of Cygwin32. The hardest part of all ports has been support for dynamic linking (HP/UX has been a major irritant on that score).

Basically, though, if you have a decent C++ compiler and GNU Make, you'll probably have little trouble with Icarus Verilog compiling for you. There is a FAQ page that shows some common problems and their solutions.

LJ: What are some other open-source EDA tools that Icarus users might be interested in?

SW: Well, there is certainly the gEDA tool suite (http://www.geda.seul.org/). And this page has links to a variety of other interesting EDA tools for Linux.

GTKWave is also a worthwhile waveform viewer. I always recommend it as a viewer that works with the VCD output from Icarus Verilog. The Electric VLSI Design System (http://www.staticfreesoft.com/) is pretty interesting, and, by the way, it does work under Linux/alpha.

The big list of EDA stuff is Open Collector, http://www.opencollector.org/. All kinds of nifty stuff can be found by browsing there.

LJ: Do you have any other HDL language support or features in mind for Icarus?

SW: Not at this time. I expect to keep up with the Verilog standardization process and that should be enough for one person. Even if I, by some miracle, “catch up” with the complete language, there will be all sorts of other aspects of compilation behind the language front end that warrant plenty of attention.

If I get bored, I might look into doing place-and-route. I need to figure out how to do that without getting too vendor-specific, though.

LJ: What would you ultimately like to see happen with Icarus?

SW:The same thing that happened to gcc, more or less.

If someone decides to pay me a lot of money to keep working on Icarus Verilog, that would be interesting, too.

LJ: What aspects of the project do you need help with?

SW: Code generators! I'm putting a lot of effort into cleaning up the API that code generators use. In my ideal world I provide a few core code generators, then let contributors do all the code generators for the various Netlist formats and simulation engines out there. This is much like how gcc works.

Regression tests! These are often scraped from bug reports, though I sometimes need to write a specific test for a feature I'm working on. The problem with me writing the tests is obvious, though.

LJ: What role do you think open-source will play with EDA tools?

SW: I don't really know. I'm not much of a sage, I'm afraid. At the very least, it should raise the minimum standards for the proprietary tools.

Also, I think open-source tools have the potential of protecting legacy designs. You can archive the source to your tools along with your design, but archiving proprietary tools seems pointless.

LJ: The mythological character Icarus had an unhappy ending. What is the significance of the name for you?

SW: You'd be surprised how few people realize this; “Icarus? how do you spell that?”

Besides the flying connection (I am a pilot, and, yes, I use FAA-approved feathers) it carries a connotation of more enthusiasm than sense. After all, I am officially a software engineer not a hardware designer. Okay, so maybe I can work an oscilloscope and logic analyzer, and I have been known to solder wires onto pins of 160-pin packages (adjacent pins), but the reality is I'm a software engineer flying too close to the sun. I've been told many times that a Verilog compiler is more work than I realize. I've been told that by hardware engineers. But not recently. I know what my limitations are supposed to be, even if I choose to defy them.

LJ: Can you tell us about the logo?

SW: Certainly. Steve Wilson can fill in more details, but basically it was drawn by a retired graphic artist, Steve's uncle. The artist, Charles Wilson, donated the design for the purpose of representing Icarus Verilog, and I appreciate the contribution. It's been used thusly ever since.

So you see, this Open Source Movement has a reach even beyond computer software.

Michael Baxter has been working in computer technology since he was nine, imprinted by a 1969 viewing of 2001: A Space Odyssey. He is an experienced computer architect, system, board and FPGA logic designer. Michael holds ten United States patents in computer architecture and logic, plus four patents as a coinventor. His Linux-related interests include open-source Verilog and EDA tools, Python, automated source code generation, concurrency and hardware issues.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Remote Sensing with Linux

Mark Lucas

Issue #82, February 2001

One company takes the initiative and saves time and money using Linux and Beowulf cluster.

ImageLinks Inc., of Melbourne, Florida, processes large satellite and aerial images for commercial businesses. This job requires processing gigabytes of image data through computationally expensive algorithms for three-dimensional projections, image processing and complex data fusions. This article describes the conversion of ImageLinks to Linux and the resulting benefits.

In 1996, ImageLinks was allowed to commercialize previously classified government software. This consisted of over 5,000 source code files of object-oriented C++ code that was developed over a period of 15 years. The business was established on high-end SGI and Sun platforms with associated servers. We were leasing over a half-million dollar's worth of equipment, resulting in a monthly payment of over $15,000. Aside from the equipment, costs there were expensive proprietary-software licenses for compilers, tools and libraries. At the time, even memory upgrades had to be purchased through the vendor at a high cost so as not to invalidate our maintenance agreements.

Figure 1. Screen shot of the bWatch program of the ImageLinks Cluster. The red lines indicate communication between nodes and demonstrate that the process is CPU-bound.

Several of the technical staff were already using Linux on home machines, and we wondered what would be involved in porting our production software. After a discussion one day at lunch we decided to stop by the local computer parts store, purchased what we needed on the company credit card and started a backroom operation to port all of the code.

We installed Red Hat 5.2 and began the porting process. Over the next couple of months Dave Burken and Ken Melero passed the project back and forth, finding and correcting platform dependencies. At the time, the main roadblock was heavily templated code that the compiler didn't handle correctly. With the release and installation of Red Hat 6.0, the GCC compiler was able to handle the templates directly, and the porting effort was completed rapidly.

Our initial assumption was that a Linux port on Intel platforms would result in a much more cost-effective solution, but we didn't believe that the performance would match the higher-end workstations. Fortunately, we were wrong on the second assumption. The first indications that we would gain significant performance improvements on the Linux platforms were observed with the compilation times. A “make World” to compile all of the source code for our software would take anywhere from 10 to 12 hours on the SGI Indigo 2s. This same compilation was completed in less than two hours on a dual-processor Pentium box—even more telling was the size of the executables that was generated. The output from the gcc tools was generating executables that were approximately half the size of the proprietary compilers. This was an indication of the superior code optimization that was one of the many benefits of the open-source development tools. This performance was quite evident when we deployed a couple of test Linux machines into production. The most extreme example was cross-sensor image fusion runs.

Cross-sensor fusion products are made by combining different classes of satellite images in order to create a new product. For example, we often combine high-resolution black-and-white imagery with low-resolution multispectral (color) imagery. The images are typically acquired from different points of view, at different resolutions, scales and times. All of these factors are taken into consideration as complex transformations are performed in three-dimensional space to project from the satellite image to an internal three-dimensional model of the space. Once this is performed, intelligent resamplers traverse the three-dimensional model to combine the pixels into the desired map projection and scale. This involves the processing of gigabytes of digital image data through complex image processing and three-dimensional transforms. It was not unusual for some of these production runs to take a weekend of processing on the proprietary workstations. With the Linux machines, we have observed almost an order of magnitude increase in performance on these fusions. This dramatic increase is due to the higher performance of commodity hardware coupled with optimized code from the software tools.

The next major benefit came from applying Beowulf clusters to our production runs. Beowulf clusters can be simply explained as a bunch of computers linked together with commodity networking for a cost-effective supercomputing solution. Most installations use Linux boxes with optimized kernels that are linked together with Ethernet communications on a local network. One node is designated as the master node controlling the scheduling of the slave nodes and all communications with the outside world. In the past, supercomputers required the software to be handcrafted for the specific architecture of the supercomputer. Recent advances in parallel libraries such as PVM and MPI have made this task much more generic. With these libraries the programmers

can identify areas of the code that can be made parallel. The libraries then take care of the details of mapping it to the super computer architecture. Fortunately, our algorithms are extremely CPU-intensive and coarsely parallel. In other words, the codes are dominated by floating point mathematical computations, and the problems can be split into parts that don't require significant communication between the processors. Our implementation involved segmenting the imagery into tiles and passing them out to different boxes for processing.

We built a 14-node cluster, wired PVM into our code and observed linear scaling in performance as we added processors to the cluster. A trace of the execution shows that there are brief periods of communication data passing to the nodes, then the nodes spend a considerable amount of time performing the necessary calculations. This turns out to be an ideal situation for the application of clustering—to go faster and push more data we can just add more processors. With the Beowulf cluster in place, the complex cross-sensor fusion jobs dropped another order of magnitude. Jobs that took a weekend on the proprietary machines, which were reduced to hours on a single Linux machine, can now run in minutes through the Beowulf cluster.

In addition to performance and cost, also our business has witnessed additional benefits including improved stability, documentation and rapid software updates.

Mark Lucas is the chief technical officer of ImageLinks Inc. He is the founder of remotesensing.org, which promotes open-source development of remote sensing and geographical information systems software. He has a BS in Electrical Engineering, an MS of Computer Science and is a retired officer from the United States Air Force.

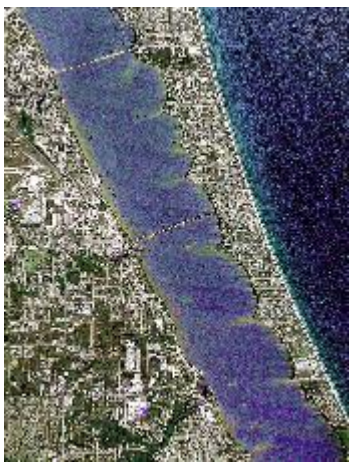


Figure 2. A section of a fused satellite image of Melbourne, Florida. This image was formed with the color from Landsat 5 combined with the 5m spatial resolution of the Indian IRS 1C satellite.

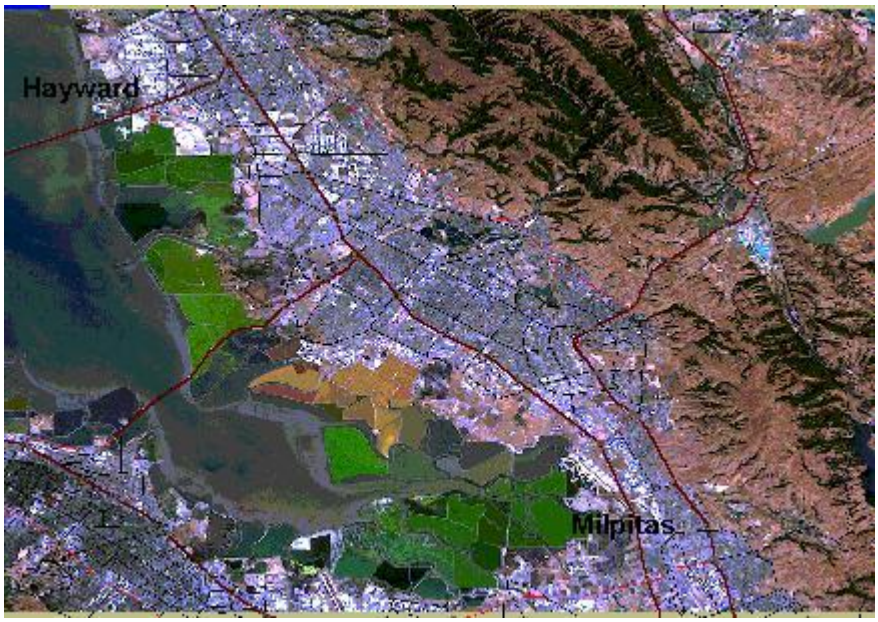


Figure 3. Composite image of the Milpitas California area. Multiple satellite images and vector layers are fused together to make the product.

Figure 4. The ImageLinks Beowulf cluster consists of 12 nodes, RAID disk drives, 100BT Ethernet Switch and power conditioning.

Figure 5. Four 650MHz Pentium III nodes with 384 Meg of memory in a 4U rack mounted unit.

Figure 6. Jeff Largent testing the nodes before they are mounted in the rack.

PocketLinux Gives Jabber Its First Hand(held)

Doc Searls

Issue #82, February 2001

“The Next Bang” prophecy fulfilled.

The cover story last September (Issue 77) was “The Next Bang: The Explosive Combination of Embedded Linux, XML and Instant Messaging”. A story with a title like that carries the burden of prophecy, so a follow up is obviously in order at some point.

As I write this, now is last November, and the buzz is still bigger than the bang, though no less portentous. *Linux Journal* gave Jabber—the XML-based, open-source instant messaging system—an Editor's Choice Award in the Communications Tool category. *Linux Magazine* named it Best Collaboration Tool. Tim O'Reilly talked up Jabber at the O'Reilly Open Source Convention and has since joined the Jabber.com board of directors. (Disclaimer: I'm on the Jabber.com advisory board, which involves equity in the company.)

Jabber.com has partnering and other kinds of deals going with VA Linux, Red Hat, Collab.net, Diamond Technology Partners and other parties. The VA Linux agreement, for example, will work Jabber instant messaging into SourceForge, as well as the Open Source Developers Network (and its various sites).

But the first place Jabber is showing up in the embedded “space” is a small one, thanks to its adoption by Transvirtual's PocketLinux as an XML transport between handheld devices. Before getting into that development, let's take a closer look at Jabber's own progress.

Jabber is fundamentally about server software and is maturing rapidly. Jabber.org has released version 1.2, which Jabber creator Jeremie Miller calls “a generational leap forward in server technology and architecture”. The big changes, he writes, are “component-based architecture and...pervasive flexibility and extensibility.” Here are some of the details:

> The new architecture is very characteristic of a typical component architecture. The core components are a client socket manager, server socket manager, session manager, file-based xdb (Jabber's interface to the file system and to databases such as LDAP—Ed.) and dns resolver. Each component is loaded and managed by the Jabber dæmon process, jabberd. Additional components can be added easily, such as protocol transports, services, agents and other utilities (note that these are not included in the core server release). These components are all assembled together by a master configuration file that acts almost like a programming environment, enabling the administrator to assemble and structure the components in a wide variety of ways. The new architecture is also very networkable, allowing any component to be removed and placed elsewhere on the network, duplicated or shared by a variety of components. This new architecture is designed to allow the server to be molded into a variety of environments and be used all the way from devices to large server farms. It's important to note that this architecture has been designed from the ground up to support operation in a server farm, with work progressing on a poll-based socket manager (tested up to 40,000 concurrent active users already). Also related is work being done on a simple name-based hashing extension to distribute load for a single domain across multiple components or servers.

He continues:

> Through the component architecture, any piece of the server can be transparently replaced by an external script or application. The jabberd process knows how to execute components as well as communicate to components via a tcp socket. With only a small Perl or Python script, or a simple Java application, additional functionality can be added to the server just about as easily as a web server can be extended via the CGI interface. All data access (vCard, roster, off line messages, etc.) can be handled by any component and authentication and registration requests can be redirected anywhere.

Jabber.org doesn't do clients, leaving that up to other groups. But, the clients it supports include all of its own, plus ICQ, MSN, AOL, Yahoo, IRC and others, through server transports written for each. Jabber Central, http://www.jabbercentral.com/, lists 14 clients, including:

- six for Windows
- four for Linux
- one for Newton
- one for Mozilla

- one for Mac
- one for Java

SourceForge lists 48 Jabber-related projects, including clients, development libraries, Perl modules, and bots one would expect to grow around an open-source instant-messaging service. Most Jabber projects are pre-1.0. According to voting at JabberCentral, Gabber is the most popular client. It runs on GNOME.

As of October 12, 2000, Jabber.com counted 50 active, open-source projects, with 17 already available. Subprojects included Palm, Java and HTTP clients, plus integration of the Jabber server with Sun's Java 2 Platform Enterprise Edition (J2EE) and Sun's iPlanet LDAP directory server.

But what makes Jabber most interesting, from an embedded perspective, is outside the realm of what we might call “classical” instant messaging. This is because Jabber may be best conceived not just as an “instant messaging system” but as a very efficient and scalable way to transport XML streams in real time. In the words of Andre Durand, Jabber.com founder and general manager, Jabber is “an XML router”.

The first company to put this aspect to use is Transvirtual (see Doc's interview with Tony Fader and Paul Fisher of Transvirtual in the January 2001 issue of *Embedded Linux Journal*), which has been active in the embedded space since it was founded in 1996. They recently developed the PocketLinux development platform, which is designed for building communications infrastructure for transporting and displaying XML documents, primarily on handheld devices. Tony Fader, Transvirtual's VP marketing, says PocketLinux is an “end-to-end system platform that allows you to write your applications in XML and Java and then run them on Linux”. It includes a small-device implementation of the Linux kernel, plus Kaffe—Transvirtual's open-source Java implementation.

Kaffe is “really a superset of the Java specification”, Fader says. “There are extensions to our abstract windowing toolkit. We have an integrated frame-buffer graphics library. We have extensions for supporting XML. We have extensions down into Linux for implementing things like video for Linux and MP3-playing and so forth.”

What Transvirtual found in Jabber was a perfect way to transport XML streams to and from small devices, especially upcoming generations of handhelds, including PDAs and mobile phones. The two development teams met at last August's LinuxWorld Expo, where the Transvirtual guys were drawing mobs of attendees by showing off PocketLinux on VTech's Helio PDA.

Says Fader, “The Jabber guys kind of tracked us down at LinuxWorld and requested that we investigate working together. We were swamped, so we didn't get around to looking at their technology until later. When we did, we found it to be a perfect match. Their technology really meshes well with what we're doing. There are other messaging systems, but they're not XML-based, or they aren't designed for instant messaging per se.”

The result is less a codevelopment effort than a vast expansion in the scope of web development. Standard PDA applications—calendar, address book, notes, task lists—no longer need to care mostly about syncing up with a PC. “The idea...is to make the apps network-aware”, says Transvirtual senior developer Paul Fisher. “We're moving to a world where you're constantly connected. That's a good thing. If people need to reach you, they can. If you need to reach somebody else, you can. Applications will presume that you're already connected to the Internet. So they can always grab or send the data they need.”

The application would essentially be written for the Jabber server, so fresh information (a change in the calendar, for example) can be pushed down to PDAs, desktops and cell phones over XML streams. The applications themselves wouldn't be locked into a hardware platform, bypassing the familiar morass of incompatibilities between all these devices. Communicating and rendering XML data are about the only requirements.

At this writing, the Transvirtual and Jabber people have collaborated on one demonstration project so far. It involves two Compaq iPAQs equipped with 802.11 wireless LAN cards, exchanging XML streams using Jabber as the transport.

“It's still early”, Fader says. “I would like to point out that not one medical app has been written for this framework yet, not one logistics app, not one shipping, manufacturing or warehousing app. The developmental paradigm here is to be able to write an application that can be fitted to any number of different sectors using this license-free framework, and this framework allows for the rapid development of cross-platform applications. It's extraordinarily compelling.”

Resources

Doc Searls is senior editor of *Linux Journal* and a coauthor of *The Cluetrain Manifesto*.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Andamooka: Open Support for Open Content

David Sweet

Issue #82, February 2001

Open-source software development provides an inspirational model for books.

A little over a year ago I began writing a book about developing KDE applications. The KDE project was introducing many new APIs and subsystems into the development code that would eventually become KDE 2.0, and developers would need documentation to keep pace with all of this new technology. But how could a book, which needs to be written, edited, reviewed and printed (with some of these steps occuring several times), be made available before the software about which it is written changes? Half of the solution was to enlist as contributing authors several talented developers who were designing and building the new technologies and to keep us all working in parallel from the then-current code base (the "bleeding edge" code). The other half, use of the Open Publication License, has lead to the development of Andamooka and the concept of open support.

The goal of Andamooka (a web-based reader support system, http://www.andamooka.org/) is to keep a book current and correct through the use of coarse-grained community annotation of the on-line text. Annotation is public and is itself open content, and so can be redistributed along with the original text in, ideally, more accurate and up-to-date annotated versions of the text.

Open Content

Open-content, using the term generally, is any content (typically textual, but not necessarily excluding sound, images and other types of "content" you might wish to consider) that can be freely modified and redistributed. If this reminds you of open-source software, it is, because that was its inspiration.

Reasons for attempting to apply the open-source software development model to open content are described in "Open Source Content Development" by David Wiley (opencontent.org/bazaar.shtml), "Why You Should Use the GNU FDL" by

Eric Raymond (www.onu.org/philosophy/why-gfdl.html) and "Do Open-Source Books Work?" by Benjamin Crowell (www.lightandmatter.com/article/article.html), but allow me to try to summarize. First of all, the open-source software development model:

- Puts software in the hands of, more or less, whoever wants it because the software is free.
- Creates more niche software because open-source software may be freely modified.
- Produces more robust software because the software gets tested by many different users in many different situations.
- Produces software with features that are in demand because lots of feedback is received from users and, often enough, users contribute by doing some development.

These four items can potentially translate well into open-content development.

Writers can reach more readers by giving their work away—and, indeed, writers want to be read. In the special case where the content is documentation for open-source software, it becomes imperative that the software documentation allow itself to be modified and redistributed so that new documentation can be derived from old documentation when new software is derived from old software.

We can consider content to be robust if it is accurate and comprehensible. The levels of accuracy and comprehensibility of content increase when reader feedback is incorporated into new versions. In other words, an open-content development model presents the image of many readers-as-editors, each focusing on the aspects of the work that interests them.

Finally, if there are many readers of a work, there will be lots of feedback that points out what is missing. This gives the author (or authors) the information they need to plan new versions of the work. If readers are welcome to participate in the authoring, they might write and submit corrected or new portions of the work, thus distributing the workload and potentially creating something better than one person (or a small group of people) could have.

Open content licenses include the Open Publication License (www.opencontent.org/openpub), the GNU Free Documentation License (www.gnu.org/copyleft/fdl/html) and the Linux Documentation Project boilerplate (www.linuxdoc.org/manifesto.html). Each of these allows the work to which it is applied to be modified and redistributed. The first of these has provisions for placing restrictions on the distribution of the work in printed format. Such a restriction could help a publishing company recoup the cost of

creating the book (a process that involves many person-hours of labor and, of course, a lot of paper). I personally recommend that if companies choose to include this restriction when a book is initially published that they later remove it, for example, after development and other costs are recouped or after a predetermined period of time has elapsed. Allowing the creation of paper versions of a work (i.e., excluding the optional restriction) might help it to reach people in markets where it would not normally be distributed.

Open Support

It should be clear by now that open-content licensing has the potential to produce high-quality content. Today there are already many open-content books available on the Web, such as *Grokking the GIMP* by Carey Bunks; *GTK+/ GNOME Application Development* by Havoc Pennington; *DocBook: The Definitive Guide* by Norman Walsh; and, Leonard Muellner and; of course, *KDE 2.0 Development* by David Sweet, et. al., but authors could do much more than simply post the text of their books if only the proper tools were available.

The Andamooka web site offers some of these tools, for free, to authors. Authors can post their book for on-line reading and annotating and for download in various formats. They can run announcements, news, questions, etc., and offer other support materials, such as source code or related documents to users. Using these tools, an author can offer support to his/her readers by interacting directly with them and by providing a forum in which readers with a common interest—after all, they're all reading the same book!—can interact. This situation differs from a less-structured public forum (like a usenet newsgroup, for example) in that the book can facilitate interaction by providing the readers with a common language in which to discuss the subject as well as a framework around which to organize their discussion.

The heart of Andamooka is annotation. The full text of each book is displayed with annotation at the end of each section. This is how the structure just discussed is realized. Annotation may be added and viewed by anyone visiting the site. The annotation is submitted under the Open Publication License so that it may be redistributed along with the book. Readers can read an annotated version of the book on-line or download it to read on their computer or to print out at home.

When collecting a large amount of information, as the community annotation for a book might become, one has to consider ways to maintain and encourage quality. Andamooka has an experimental moderation and points (or Karma, à la Slashdot) system based on the SlashCode system. A score is assigned to each piece of annotation based on random, occasional polling of readers. When polled, a reader can choose one of three ratings (let's call the rating r): good

(r=3), neutral (r=2) or bad (r=1). For the purpose of ranking the annotations for viewing, attempting to put the “best” at the top, a score (call it S) is assigned to each article. If we call N the number of times an article has been rated, then the score is: S=SQRT(N)*AVG(r)/STDDEV(r). In plain English, this means that an annotation is rated higher if:

- More people have rated (and, presumably, read) it (SQRT(N))
- People have, on average, given it a better rating (AVG(r))
- People are in greater agreement about the quality of the annotation (1/STDDEV(r))

To encourage quality submissions, users are given points—or “Karma”--based on how the annotation they write is rated by the community and, possibly, based on how much they contribute and how recently they have contributed. These features have yet to be fully implemented.

The hope is that this system will serve as a community editing, offering a way for readers to sort out the best annotation. As Andamooka is in its infancy, I cannot say whether this system is successful or not.

Open Development

The next step is the creation of an open-development system. The model, as it is discussed today, derives from open-source software development (as mentioned above) but has some different issues to contend with. Those interested in open development would like to create a book, or other work, by having many contributors plan, write, edit, format, etc., in hopes that the workload will be distributed, the work will be of high quality and everyone will enjoy what they are doing (since it is volunteer work). Similar goals have been achieved in the open-source software world, so what's different about open content?

Authors may not be as technically savvy as software developers (note that by “authors” I don't necessarily mean authors of technical books). So, unlike software developers, they can't necessarily make use of software-project management tools (for example, CVS) that are difficult to install, maintain, and/or used to develop a book, however capable those tools might be. Thus, there is a need for user-friendly project management and collaboration tools. The TWiki web-based application (http://twiki.sourceforge.net/) and its Wiki Wiki kin are good examples of these kinds of tools, but they need to be expanded upon to be appropriate for open-content development. The members (myself included) of the FreeBooks Project (http://freebooks.myip.org/), founded by Benjamin Crowell, are exploring designs for the needed tools and methods for managing this type of project (for example, we are testing a method of

incorporating DocBook document collaboration into TWiki). In fact, this exploration process has been organized around writing a free book about writing free books. (You are encouraged to participate. Please visit the web site for details.)

One also has to wonder whether an open, community-development model for content will attract contributors the way similar software projects do. So far, several people are contributing to the FreeBooks Project, and translations of *KDE 2.0 Development* into five other languages are underway. These numbers may not be as impressive as those used when discussing open-source software, but these projects have only just begun, and I consider the level of participation to be very encouraging.

Conclusion

Andamooka is a first step toward full open-content development. The books we see on the site now, and probably those we'll see in the near future, were written in a closed manner but are being made available under open licenses that allow modification and redistribution. By making use of Andamooka, authors can offer readers direct author support and the support of rest of the readership. Together the authors and readers can create a base of knowledge that can be culled and transformed—by any community participants—into a better and better book.

I plan to continue work on *KDE 2.0 Development* using the Andamooka system. I will encourage users to offer feedback and contribute to development by writing, editing, translating, reporting errors and suggesting new topics. This work will be released later in free, electronic editions of the book. As such, *KDE 2.0 Development* serves as a test case for open-content development.

David Sweet (www.andamooka.org/~dsweet) received his PhD in Physics/Chaos Theory from the University of Maryland. His focus has since shifted from the chaos of Christmas ornaments to the sheer noise of the securities markets. His work has appeared in *Nature*, *Linux Journal*, *Physical Review Letters* and in a few bookstores.

Archive Index Issue Table of Contents

Advanced search

Linux as a Video Desktop

Robin Rowe

Issue #82, February 2001

Rowe introduces a new article series on developing Linux multimedia applications from scratch.

For the Emmy award-winning movie *Titanic* the production studio Digital Domain used Linux on a network of more than a hundred DEC Alpha machines to render the special effects. However, they used 350 SGI machines running IRIX and a hundred DEC Alphas running Windows NT for the artistic aspects of the production. Linux was judged not-ready in 1997 for the video desktop. What's developed with Linux as a desktop operating system for video production in the last few years?

For Linux to be a player in the realm of television and motion picture production it must have stunning graphics capabilities, high-performance disk I/O with support for the very large file sizes video needs, integration with analog and digital video input/output devices, and the actual video applications themselves. As these capabilities are becoming available on Linux, the companies that develop video applications (including the company I work for) are looking at Linux in a new way.

Because Digital Convergence seems like too much of a mouthful for a title, and Linux Multimedia too lame, we'll call the column Media/GFX. GFX is an abbreviation of “graphics” used in television news rundowns and scripts, a term I picked up years ago as technical director for news production at an NBC television station. Media/GFX will tell about our travails with Linux over the next year as we install multimedia applications and make progress on the programming front and will discuss what it is like to develop multimedia applications for Linux from scratch. My own area of expertise is video software design using C++ and Java, and that's always interesting (and frustrating) on any platform. Video applications are always some of the most challenging because of the demands they make on the CPU, disk and I/O systems.

Companies that develop software for their livelihood need to be practical in the operating system they choose. It needs to be popular, capable and not too hard to develop software for. Every few years that platform tends to change. For example, during the past four years my software development has been mainly on Windows NT, with some code ported to Solaris and Alpha for servers. A major motivation to use Windows is the vast array of relatively inexpensive hardware available for it, including MPEG-1 and MPEG-2 video compression cards. Before my switch to WinNT it was Sparc20 Solaris with a Parallax Motion-JPEG card, and before that a simple SGI Indigo running IRIX. If you are a video software developer it is best not to get too attached to an operating system.

The Macintosh was perceived as the graphics platform of choice for many years but has been eclipsed by the sheer popularity of Windows. Windows 98 and Millennium Edition versions are the dominant platforms for games utilizing high performance graphics. However, stability and security concerns with DOS-based Win9x tend to preclude its use in high-reliability broadcast scenarios. For the same reason, Win9x is broadly prohibited for Department of Defense users. The DoD is a major consumer of graphics systems, usually WinNT or Solaris. Windows NT/2000 is also a major player in video workstations. An OS design influenced by VMS (which was also created by the same architect), WinNT/2k is more like UNIX than the DOS-based Windows versions, even to POSIX compliance. In the high-end, SGI with its IRIX flavor of UNIX has long dominated motion picture production video workstations. However, WinNT/2K has made inroads.

Earlier this year SGI announced they are moving to Linux in a major way. The big news is graphics, the area SGI has built its reputation upon. In collaboration with video chip manufacturer NVIDIA, SGI Intel-based Linux workstations models 230, 330 and 550 can, according to SGI, draw over 17 million triangles per second and up to 540 megapixels per second. Their VPro graphics subsystems are 32MB or 64MB DDR AGP 4X boards. Perhaps like me, you're not ready to rush out and buy an SGI workstation. You don't have to. The graphics improvements have been made open source and rolled into XFree86. The graphics capabilities of Linux have taken a quantum leap forward!

Another area of video capability where SGI is making a significant contribution is file systems. A DV IEEE-1394 Firewire video transport stream moves at about 30MBs (typically advertised as just 25MBs, but that's counting video only, not audio). At that bit rate you will bump up against the 2GB file size limit in Ext2 in less than nine minutes. To contain one 22-minute television program you need to create a 5GB file. For a 100-minute movie, make that 23GB.

XFS is a high-performance journaled file system capable of handling files as large as 9,000 petabytes that would hold a DV file lasting 837 centuries. XFS has

been ported from SGI IRIX to Linux and was released in September in beta. Journaled file systems treat disk activity as database transactions and are able to roll back to a stable configuration if interrupted by a crash or power-down. No fsck. XFS isn't the only high-performance Linux file system available. Other journaled file systems (not from SGI) include JFS, ReiserFS and Ext3FS. Journaled file systems don't have to handle large files. In fact, Ext3FS, built upon Ext2, isn't suitable for large video files because it inherited the same 2GB limit.

SGI GLX is the glue code that connects the board-level OpenGL graphics language with the X Windows System. The GLX Client Library presents the popular OpenGL API to applications, performs indirect rendering via X11 network protocol, and high-performance direct rendering where the client display and server are the same machine. The Precision Insight implementation is called the Direct Rendering Infrastructure (DRI) and uses GLX, Mesa 3.1, XFree86 4.0 and Linux kernel modifications. The work of porting GLX to Linux was carried out by Precision Insight, sponsored by SGI and Red Hat.

OpenGL is a platform-independent scene description language, implemented in the hardware of most high-performance graphics cards and even in cheap PC cards costing less than $100 today. It is the most widely available 3-D API (Linux, Mac, Win32 and others), beating Microsoft Direct3D and 3dfx Glide. OpenGL enables 3-D application programmers to get their software to work on any hardware without writing their own drivers for every 3-D board out there. Being a standard 3-D API of course means that OpenGL can be used for 2-D as well.

A game designer uses OpenGL to describe what to draw as a set of graphics primitives (shapes in space) and texture maps (images). Having hardware handle these 3-D calculations can vastly speed up screen rates in virtual reality interactive games such as *Quake* (http://www.quake3arena.com/).

Figure 1. Screen Shot from Quake III

SGI's reference implementation of the OpenGL API is widely used to develop hardware drivers for their systems. Mesa, a 3-D graphics library with a work-a-like API similar to that of OpenGL, is the glaring exception. Mesa (http://www.mesa3d.org/) was developed independently to circumvent the OpenGL commercial license. SGI has since made OpenGL, GLX and XFS open source. SGI Open Inventor is a popular object-oriented 3-D toolkit based on a 3-D scene database, also now open source. Open Inventor is built on top of OpenGL and supports PostScript printing. See the SGI open-source web site at www.sgi.com/developers/oss/.

With high-performance graphics only now becoming available, graphics applications for the Linux platform are still catching up with their SGI IRIX and Windows NT counterparts. The GIMP, of course, is highly regarded as a replacement for Photoshop. But, did you know you can use GIMP to overlay graphics in Broadcast2000 [see Adam Williams' “Moviemaking on a Linux Box? No Way!” in the January 2000 issue—Editor], a Linux video nonlinear editor? Broadcast2000 manipulates uncompressed 720x480 video captured using a Video4Linux-compatible video card such as the inexpensive Haupaugge WinTV or cards from Linux Media Labs. Audio is 48K stereo sound from an OSS-compatible audio card. Starting with kernel version 2.2, Linux supports the DV IEEE-1394 Firewire protocol.

Many graphics programs are available for the Linux platform in addition to the GIMP. Blender is a popular animation package to create stunning three-dimensional graphics. Houdini was the first major commercial 3-D animation package available on Linux. Ported from SGI IRIX, Side Effects Software's Houdini has been used extensively for effects animation at Digital Domain and other major studios. It was used for effects in *Titanic* and in *X-Men*. At $17K US, Houdini is not free, but a 30-day evaluation version is available.

During the coming months we will investigate every Linux multimedia-related item we can:

- Video4Linux with a Haupaugge WinTV card
- 2-D and 3-D graphics and video programming using C++, OpenGL and Java
- MPEG-1 and MPEG-2 players and encoders
- DV players and editors, IEEE-1394 Firewire
- AVI video
- Quicktime video
- MP3 and alternative audio formats like OGG
- Streaming with Real and Quicktime
- HDTV
- Houdini, the software used to create the special effects for *Titanic*

- Animation, rotoscoping
- How Linux software applications compare with running the same on other OSes
- More codec stuff
- Web authoring (e.g., Amaya)
- Game graphics, using SVGA and so forth
- Hardware (e.g., IEEE-1394 OHCI Firewire, WinTV, ATI All-in-Wonder)

An expert with Linux might be able to cover these easily but not with the same point of view. With almost no experience with Linux personally (although quite a bit with Solaris and Windows) we can expect to make some of the dumb goofs that any Linux newbie would make. By going through this process from the start, we can learn together.

We are right at the start of our Linux multimedia installation. A few days ago a new Maxtor 20GB 7200RPM hard drive was still in the box next to my desk waiting for a fresh install of Win98, Win2k and Debian Linux on it. We're creating a triple-boot system because we still have our regular Windows development work to do for our existing products. Also, it is our perhaps naïve plan to develop the same applications across Windows and Linux from a single-code base. That will be some trick given the differences between the X and Win32 GUI APIs.

Our PC contains a WinTV card for watching/capturing television and a PyroDV IEEE-1394 Firewire card to interface with a digital camcorder. When we get all that working right we will add an ATI All-in-Wonder and maybe a Compaq iPAQ handheld that we want to do some multimedia development with. (We too saw the PocketLinux iPAQ running at the San Diego USENIX convention, very cool. See http://www.handhelds.org/.)

Linux continues to excel as a network server. The BBC uses Linux to serve web pages, RealMedia and their digital text service. Victoria's Secret relied on VA Linux servers to stream their Cannes 2000 Fashion Show, one of the largest live webcasts in Internet history, more than two million viewers worldwide. With support from SGI, NVIDIA, Red Hat, Side Effects Software and many others, Linux is expanding beyond network servers and animation render farms into desktop video production. Next month we'll boot our new system and start finding out just how good the Linux is becoming as a video desktop.

Robin Rowe (Robin.Rowe@MovieEditor.com) is a partner in MovieEditor.com, a technology company that creates Internet and broadcast video applications. He has written for *Dr. Dobb's Journal*, the *C++ Report the C/C++ Users Journal*, *Data Based Advisor* and has had many papers published in conference proceedings. His software designs include a client/server video editing system in use at Manhattan 24-hour broadcast television news station Time Warner New York One and associated web site http://www.ny1.com/, and an automated television news monitoring system developed for DARPA and the Pentagon. He has taught C++ at two universities and designed video software in Fortune 500, DoD and academic environments. You can reach him at Robin.Rowe@MovieEditor.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Loadable Kernel Module Programming and System Call Interception

Nitesh Dhanjani

Gustavo Rodriguez-Rivera

Issue #82, February 2001

Loadable Kernel Module Programming and System Call Interception

Modern CPUs can run in two modes: kernel mode and user mode. When a CPU runs in kernel mode, an extended set of instructions is allowed, as is free access to anywhere in memory and device registers. Interrupt drivers and operating system services run in kernel mode. In contrast, when the CPU runs in user mode, only a restricted set of instructions is allowed, and the CPU has a restricted view of the memory and devices. Library functions and user programs run in user mode. Kernel and user mode together form the basis for security and reliability in modern operating systems.

Programs spend most of their time in user mode and switch to kernel mode only when they need an operating system service. Operating system services are offered through system calls. System calls are “gates” into the kernel implemented with software interrupts. Software interrupts are interrupts produced by a program and processed in kernel mode by the operating system.

The operating system maintains a “system call table” that has pointers to the functions that implement the system calls inside the kernel. From the program's point of view, this list of system calls provides a well-defined interface to the operating system services. You can obtain a list of the different system calls by looking at the file **/usr/include/sys/syscall.h**. In Linux, this file includes the file **/usr/include/bits/syscall.h**.

Loadable modules are pieces of code that can be loaded and unloaded into the kernel on demand. Loadable modules add extra functionality to the kernel

without the need of rebooting the machine. For example, it is common in Linux to use loadable modules for new device drivers. The alternative to loadable modules is a monolithic kernel where new functionality is added directly into the kernel code. Monolithic kernels have the disadvantage of needing to be rebuilt and reinstalled every time new functionality is added.

Kernel programming can be difficult not only because of the intrinsic complexity but also because of the long debugging cycle. Debugging an operating system may require installing a new kernel and rebooting the machine in every cycle. We strongly recommend using loadable modules in kernel development because a) there is no need to rebuild the kernel or to reboot the machine more often than necessary; and b) since the end user does not need to replace/rebuild the existing kernel, the user is more likely to install the new functionality.

Loadable module support within the Linux kernel facilitates the interception of system calls, and this feature can be taken advantage of as described within the examples below. As a note, it is assumed that the reader is familiar with C programming.

1. System Calls, an Introduction

Operating systems provide entry points through system calls that allow user-level processes to request services from the kernel. It is important to distinguish between system calls and library functions. Library functions are linked to the program and tend to be more portable since they are not bound to the kernel implementation. However, many library functions use system calls to perform various tasks within the system kernel. To illustrate, consider this C program that opens a file and prints its contents:

```
#include <stdio.h>
int main(void)
{
    FILE *myfile;
    char tempstring[1024];
    if(!(myfile=fopen("/etc/passwd","r")))
    {
         fprintf(stderr,"Could not open file\n");
         exit(1);
    }
    while(!feof(myfile))
    {
         fscanf(myfile,"%s\n",tempstring);
         fprintf(stdout,"%s\n",tempstring);
    }
    exit(0);
}
```

Within the program, we used the **fopen** function call in order to open the **/etc/passwd** file. However, it is important to note that fopen is not a system call. In fact, fopen calls the system call **open** internally in order to do the real I/O. To get a list of all the system calls invoked by a program, use the **strace** program.

Assuming you have compiled the above program as a.out by running gcc example1.c, running strace like : **strace ./a.out** will allow you to see all the system calls being invoked by **a.out**.

The kernel switches to the user-id of the process owner invoking the system call. So, if a regular user were to run the above program, with **/etc/shadow** (which is not readable) as the parameter to fopen, the open would fail and so would fopen, causing the if clause above to translate to true, thus printing the **Could not open file** error message.

2. Intercepting System Calls via Loadable Modules, an Example

Assume that we want to intercept the **exit** system call and print a message on the console when any process invokes it. In order to do this, we have to write our own fake exit system call, then make the kernel call our fake exit function instead of the original exit call. At the end of our fake exit call, we can invoke the original exit call. In order to do this, we must manipulate the system call table array (sys_call_table). Take a look at **/usr/src/linux/arch/i386/kernel/entry.S** (assuming you are on an i386 architecture). This file contains a list of all the system calls implemented within the kernel and their position within the sys_call_table array.

Armed with the sys_call_table array, we can manipulate it to make the **sys_exit** entry point to our new fake exit call. We must store a pointer to the original sys_exit call and call it when we are done printing our message to the console. Source code to implement the above is as shown in Listing 1.

Listing 1. Example 2.c

Compile the program shown in Listing 1 by invoking gcc: **gcc -Wall -DMODULE -D__KERNEL__ -DLINUX -c example2.c**. This gives us our example2.o module. In order to insert this module into the kernel, do this as root: **insmod example2.o**. Now, make sure you are on the console (since **printk** only prints to the console), and run any program which uses the exit system call. For example, **ls** should print: **HEY! sys_exit called with error_code=0**.

Next, try to invoke **ls** with a file that does not exist; this should cause **ls** to call the exit system call with an argument other than 0. Therefore, **ls somefilethatdoesnotexist** should print: **HEY! sys_exit called with error_code=1**.

In order to list all the modules loaded, use **lsmod**. To remove the module, run **rmmod example2**.

3. A More Interesting Example: Intercepting sys_execve to Protect Against

"root-kits"

After a machine is compromised, malicious users tend to replace commonly used programs with trojan horses (programs that execute malicious instructions in addition to their normal functions). Packages of such trojan horses are widely distributed over the Internet and are easily accessible by anyone. Therefore, it becomes important to protect programs from being replaced by malicious users.

In order to protect against such problems, our next example involves the interception of various system calls, most importantly **sys_execve**, to check the hash of the program to be executed against a known hash present in a database file. The program is denied execution if the hashes do not match, and such an attempt is logged. One way to implement this is seen in the following steps:

1. Intercept **sys_execve** and compute the inode of the file being executed, than compare it with the inodes of the files present in the hash database. Inodes are data structures that contain information about files in the file system. Since there is a unique inode for each file, we can be certain of our comparison results. If no match is found, call the original **sys_execve** and return. However, if a match is found, compute the hash of the program and then compare it with the hash present in the hash database. If they match, call the original **sys_execve** and return. If they do not match, log the attempt and return an error.
2. Intercept **sys_delete_module**. If called with our module name as the parameter, return an error. Our module cannot be deleted.
3. Intercept **sys_create_module**, and return an error. No more modules can be inserted because we do not want any malicious module to be able to intercept the **sys_execve** described in step 1.
4. Intercept **sys_open** to prevent our hash database and log file to be opened for writing.
5. Intercept **sys_unlink** to prevent deletion of the hash database and log file.

Note that the above does not offer complete protection, but it is a simple first-step implementation. For example, a malicious user may modify kernel symbols in **/dev/kmem** or use raw device access to the hard disk, and bypass open to write to the hash database file. Also, since our implementation is only a loadable module, a malicious user can alter our **/etc/rc.d** files and stop our module from being loaded the next time the machine is rebooted. In addition, various other system calls exist that could cause our hash database and log files to be altered or deleted.

At this time, it becomes important to acknowledge the possibility of loadable module support being misused by a malicious user. For example, the **sys_execve** function call can be intercepted to invoke a trojan program instead of the one intended, and system calls such as **read** and **write** can be intercepted to perform keystroke logging. Therefore, the flexibility and power of loadable kernel modules can be misused by malicious users who may have gained access to the system. See Resources for a web site that has details of this example along with complete source code.

Resources

Acknowledgements

Nitesh Dhanjani is a graduate student at Purdue University. His interests are operating systems, networking and security. He has performed security audits and reviews for various firms including Ernst & Young LLP, and offers consulting services in his spare time. He can be reached at dhanjani@dhanjani.com.

Gustavo Rodriguez-Rivera is a Visiting Assistant Professor at Purdue University and is also software architect for Geodesic Systems. His interests are operating systems, networking and memory management. He can be reached at grr@cs.purdue.edu.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

More with Three-Tiered Design

Reuven M. Lerner

Issue #82, February 2001

Reuven shares more tips on the usefulness and limitations of three-tiered architecture—and just what is it?

Last month, we began our investigation of three-tiered design for our web applications. By separating the database server from the web application itself by means of a “middleware” object layer, we simplify the logic in our web applications. Furthermore, by adding an abstraction layer between our web application layer and our database layer, we gain the ability to use the same middleware in non-web applications, as well as the possibility of changing the back end without telling the web application.

By the end of last month's column, we had implemented a simple middleware layer that could communicate with the People and Appointments tables we created in a PostgreSQL database. This month, we will briefly look at some web applications we can develop using these objects. You will see that at no time does our web layer directly access the relational database; the SQL is all contained within the objects.

The Web Application Layer

In an ideal universe, we could create the web application layer using any language or technology we might want, communicating with the middleware layer using a universally agreed-to protocol. However, the world is not quite as advanced as we might like, and our choice of an object layer forces our hand when choosing a web application environment.

We created our objects in Perl, so we will need to use Perl to implement our web application. To avoid the overhead associated with CGI programs, and because we can get a great deal more power by tapping into the mod_perl module for Apache, we will use Mason, the Perl-based template and development application environment that we looked into last year. Each

Mason component is compiled as necessary into a Perl subroutine, which is then compiled into Perl opcodes. These opcodes are then cached in the mod_perl module inside of Apache, where they can be executed at a much faster rate than would be possible using CGI.

Adding a Person

Our first web application example will allow us to add a new person to our database. This will require two Mason components: an HTML form (which could equally well be a static form) and one which attempts to add a new person to the database. In order to accomplish this, we will use the middleware People object, which connects to the database for us and attempts to store a new row in the database. Simple versions of these two components are shown in Listings 1 and 2. These listings are too long to print here; they are available at ftp.linuxjournal.com/pub/lj/listings/issue82. The HTML form (add-person-form.html) submits its name-value pairs to add-person.html. The latter creates an instance, People, then invokes the new_person method to create a new person:

```
my $success = $people->new_person
                        (first_name => $first_name,
                        last_name => $last_name,
                        country => $country,
                        email => $email);
```

If $success is true, we know that a new person was added to the database with the arguments that we passed to $people->new_person. Otherwise, we know that the invocation has failed.

However, this is a very crude way of determining whether things have succeeded or failed; rather than present users with an all-or-nothing proposition, it would be nice to tell them what they did wrong so that they can fix the problem. If a hung database process produces the same error message as does an attempt to add a second person with the same e-mail address, it will be hard for anyone to solve the problem.

Thus, the solution is for our web application to check its inputs before passing them to the middleware layer. The more such checks we can insert into our code, and the more application-level error messages we can display, the better.

Our add-person.html component performs two basic checks that demonstrate this: It uses Mason's <%args> section to require that each of the potential arguments has been passed. An HTML form that tries to submit its values to add-person.html must provide each of the listed form elements, or Mason will refuse to honor the request and print a stack trace describing what went wrong. End users won't see this error if they make a mistake filling out the form, but you'll see it if you leave required <input> tags out.

Once our Mason component executes, we can thus be sure that we have at least received the appropriate name-value pairs. But do they contain legal values? In an “unless” statement at the top of add-person.html, we check that we received non-empty values for the four parameters that we will use in our invocation of $people->new_person. If any of them are missing, a message is displayed telling the user what is expected.

To be even safer, we also check that the e-mail address looks relatively valid. The regular expression in Listing 2 will not match all e-mail addresses, but it is good enough for the purposes of this simple example. Users who try to pass an invalid e-mail address are shown an error message that tells them what to change.

Once we can be sure that the values are relatively sane, we can then invoke $people->new_person. Notice how add-person.html manages to do all of this without ever talking directly to the database. DBI is obviously taking an active role in each invocation of $people->new_person, but that happens behind the scenes, and our Mason components don't need to concern themselves with it. This means that if the People object has been thoroughly debugged, there should not be any chance of encountering SQL errors.

Editing a Person

Now that we have seen how to add a new person to the database using our People object, let's try a slightly more difficult task: changing a person's first name, using the update_first_name method. (See Listing 3 and Listing 4 at ftp://ftp.linuxjournal.com/pub/lj/listings/issue82/ for examples.) We can only invoke this method once we have already selected an individual, which means that our editing form will have to let us do so.

While it might be tempting to let users select an entry by typing a name or e-mail address into a text field, this is prone to too many errors to be effective. Instead, we will allow users to choose from a <select> list. This removes the possibility that a user will enter an e-mail address (or another defining characteristic) for a person who might not be in our database.

We want to use a unique key to choose the person whose first name will be modified—but at the same time, it seems a bit impersonal to present a list of e-mail addresses. My solution was to go back to the People object (People.pm) and define a new method, get_names_and_addresses (see Listing 5 at ftp://ftp.linuxjournal.com/pub/lj/listings/issue82/). This returns a list of array references, where each array reference contains a name and an e-mail address. The former can be used as a unique key (and as the “value” within an <option> tag), while the latter can be used for display purposes. We can thus iterate over the e-mail addresses and produce a <select> list as follows:

```
<select name="email">
% # Iterate through the names and addresses,
  # printing them out
% foreach my $info (@names_and_addresses) {
  <option value ="<% $info->[1] %>"><% $info->[0] %>
% }
</select>
```

Allowing users to edit other user attributes would proceed in a similar way. Indeed, so long as you ensure that the user chooses a key that uniquely identifies the user, you can change any and all of its attributes using a similar type of form.

Adding an Appointment

Now that we have seen how we can use the People object to indirectly manipulate our People table in the database, we will start to look into our appointment book, handled by the Appointments object. This object allows us to add an appointment with a particular person on a particular day and time.

In order to accomplish this, we will (once again) need two components. The first component (add-appointment-form.html, in Listing 6 at ftp.linuxjournal.com/pub/lj/listings/issue82) produces an HTML form that allows users to enter a new appointment into the system, choosing a person from a predefined <select> list. (If this were an actual project, I would put the <select> list in a separate component, allowing other components to produce a menu of entries in the address book.) In addition, we have to know when the appointment starts, as well as when it ends. Once again, I prefer to have people select a date and time from <select> lists, since it removes the problems associated with time and date formats.

The following Mason code produces the three <select> lists that we need in order to have the user choose a month, day and year. By defining the @months and @years arrays in advance, we can make the code more readable, as well as update the system for future years quickly and easily:

```
<select name="begin_month">
% foreach my $month (@months) {
    <option value="<% $month %>"><% $month %>
% }
</select>
<select name="begin_day">
% foreach my $day (1 .. 31) {
    <option value="<% $day %>"><% $day %>
% }
</select>
,
<select name="begin_year">
% foreach my $year (@years) {
    <option value="<% $year %>"><% $year %>
% }
</select>
```

The second component, add-appointment.html (see Listing 7 at ftp.linuxjournal.com/pub/lj/listings/issue82), allows us to add a new entry into the appointment calendar. It checks (using an <%args> section) that we have submitted all of the required name-value fields from add-appointment-form.html. We then issue the same kinds of basic checks that our other components have done.

Are These Three Tiers?

Now that we have demonstrated how easy it is to create a three-tier web application, it's time to consider how many tiers we're really using. Does the term “three-tier” really apply here?

The term “three-tiered architecture” grew out of a dissatisfaction with another popular architecture known as “client/server”. For example, databases and web servers are both examples of modern client/server systems. Just as a client/server system typically refers to two physical computers, a three-tiered system refers to three physical computers, with each tier residing on a separate machine.

By contrast, the simple three-tiered application we have examined certainly had three layers in that there were distinct software systems that had clear goals and APIs and made it possible for the application and database layers to speak through a common middleware layer. At the same time, at least two of these layers (the web application and the middleware objects) were on the same computer without any real possibility for separation. If the web application were to become swamped with traffic, we could certainly add one or more identical Apache servers—but there is no way to put the application layer on one computer and the middleware objects on another.

So while I believe that we have now demonstrated some of the advantages of three tiers from the perspective of an application developer needing standard APIs, we have not seen a true implementation of such a system. In order to do so, however, we must have the ability to perform remote procedure calls (RPCs), such that a web application on one computer can invoke a subroutine or object method on another computer. This is possible and is getting increasingly easy with the growth of SOAP (Simple Object Access Protocol), but it brings with it a number of other problems and caveats, including the need to learn yet another transmission protocol.

While we're reconsidering how to define tiers—perhaps we should consider that the application developed does have three tiers, but that they are defined in a different way than we have considered until now. Instead of counting the tiers as (a) database, (b) middleware layer and (c) web server, perhaps we should count them as (a) database, (b) web server and (c) web browser? If we

think in these terms, then we have indeed created a three-tier architecture—but come to think of it, so has anyone who has ever written a CGI program that talks to a database.

Moreover, we can introduce additional layers of abstraction into the mix here, complicating things further. What about stored procedures, triggers and views created on the relational database? Although not a physical tier, it can certainly make life easier for the person writing either an object layer or an application that accesses the database. Indeed, stored procedures are often better than an object middleware layer, because they execute on the database and are precompiled, making them relatively speedy.

We can also execute code on the web client (i.e., inside of the web browser) using JavaScript. While I generally encourage my clients to avoid JavaScript as much as possible, this buggy, insecure language riddled with cross-platform incompatibilities is the only way to execute programs from within a web browser, rather than on the server.

So, when we have a web application that uses a relational database, stored procedures, an object middleware layer, a web application layer and client-side JavaScript divided between three computers, how many tiers do we have? It's probably still three, but the fact is that it doesn't really matter what you call it. In the end, a decent design that takes into account your project's specifications, including the need for future growth, is the right way to go—regardless of how it jibes with the latest buzzwords and techniques.

Problems with Three-Tier Design

Now that we have looked at a very simple three-tier project, it's time to look at some of the problems associated with such a design. I am not saying that three-tier solutions are inherently evil, but neither are they a panacea. Like most solutions, they are appropriate under certain circumstances. In many cases, splitting the design and implementation among several people can be easier when you divide the work into different layers, as we saw with our web-based appointment calendar. One person could write all of the necessary code, but two people could probably do it more quickly and easily, given a well-documented interface between them.

As with any engineering solution, there are always trade-offs. With a three-tiered design, perhaps the most important trade-off is time. Such an architecture takes longer to specify and design, even if it will eventually be more robust, easier to write, easier to test and easier to divide among a number of programmers.

While dividing a project into many parts might make it easier to specify and test each part, it makes integration testing all the more important and difficult. If everyone sticks to the published and agreed-to API, such testing does not have to be terribly difficult. But there are always differences between the specification and the implementation, and integration testing tends to bring these out. The more tiers in a project, the more important and difficult such testing can be.

Finally, it can be difficult and frustrating to create an object middleware layer that provides an interface to the database. SQL is not a perfect language, but it allows us to express a very large number of queries with a very small number of commands.

Removing SQL from the Mason components and forcing programmers to work with an object API, means that the programmer will be limited to a small subset of the database's power and flexibility. Every time more functionality is needed, the programmer will have to request it be added to the middleware's API. Being able to specify any database query inside of an HTML template (e.g., a Mason component) is a liberating experience for a programmer, and taking that freedom away can be frustrating.

Conclusion

Programmers designing large or complex web applications are finding it increasingly useful to adopt the three-tiered architecture beginning to replace the simpler client/server model that has been favored for the last decade. Indeed, creating three-tiered applications can often make life easier. In the end, however, you will have to decide whether this solution is appropriate for your needs or if it's overkill given your time frame and specifications.

Reuven M. Lerner owns and manages a small consulting firm specializing in web and Internet technologies. As you read this, he should be (finally!) finishing Core Perl, to be published by Prentice-Hall later this year. You can reach him at reuven@lerner.co.il, or at the ATF home page, http://www.lerner.co.il/atf/.

Archive Index Issue Table of Contents

Advanced search

Smell of Fresh-Baked Kernels

Marcel Gagné

Issue #82, February 2001

With some tips from our French chef, and a glass of wine or two, compiling is a manageable task.

Mon Dieu! François! Come out from under that table *immédiatement*! You are very lucky that our customers have not yet arrived. To see you hiding from the prospect of a kernel recompile. *Qu'est-ce que je vai faire avec toi?* Sit down and let me pour you a glass of wine to steady your nerves. There. Feeling better? *Bien*.

Log in and let me show you how easy it is. You know, François, that I have been collecting from other open-source chefs some wonderful recipes designed to make this whole kernel process that much more attractive to the palate, *non*? *Quoi*? *Ah, mes amis*. Welcome! François, what are you doing sitting there drinking? Show our guests to their seats. When you are done, bring up the 1996 Meursault Genevrieres from the cellar.

Pardon, mes amis. François and I were discussing the intricacies of Linux kernel builds. My faithful waiter was about to build his first kernel and, I must tell you, he is a little shy about the idea. It is for François and others like him that I have sought out some of the items on tonight's menu.

Building a kernel is not as much of an ordeal as it once was. Part of the reason, I must admit, is that the *need* to recompile your kernel is becoming less and less of a requirement for the average user. Out of the box, most modern distributions include support for more popular devices either in the kernel or through a collection of compile loadable modules. Still, there are times when the distributed kernel will not support your hardware. You may also need specific features that, while uncommon, are required in your environment.

This generally implies a requirement for the latest and greatest Linux has to offer. If you want to know what the latest version of the Linux kernel is, whether it is a stable release or a development version, try out the following **finger** command:

```
finger @finger.kernel.org
```

A few seconds later, the system responds with something like this:

```
[zeus.kernel.org]
The latest stable version of the Linux kernel is:
        2.2.17
The latest beta version of the Linux kernel is:
        2.4.0-test10
The latest prepatch (alpha) version *appears* to be:
        2.4.0-test11/pre5
```

This assumes, of course, that you are *currently* connected to the Internet. I see that one of our diners is putting up their hands and mouthing the word *always*. *Bravo, mon ami*.

Armed with the latest kernel information, you can now visit your favorite kernel source mirrors and download what you need. Notice that I said *mirrors*. Getting to your local mirror is pretty simple. All you have to do is squeeze your country code in between the “ftp” or “www” of kernel.org. For instance, assume for a moment that I am in Canada (country code “ca”). The addresses for the ftp and www sites would be as follows:

```
ftp.ca.kernel.org
www.ca.kernel.org
```

Similarly, if I were in France, I would use these mirrors:

```
ftp.fr.kernel.org
www.fr.kernel.org
```

If you are unsure about any of these codes, visit www.kernel.org/pub/mirrors for a complete list.

Staying on top of all this can be a chore as well. You may have to follow kernel development quite closely (perhaps daily). Pretend for a moment that you have a new card that lets you input commands directly into your system via thought transmissions (*Incroyable!*) and kernel support is currently in development. Whatever your reasons, *Darxus* provides you with a nice little Perl script to help you out. It is called **dlkern** and is available from his web site (see Resources). With dlkern, you simply tell the script whether you want the stable version of the kernel (-s option), the development or beta version (-b option), or the pre-patch alpha version (-p option). That is all. Allow me to demonstrate. In the next example, I retrieve the latest stable kernel:

```
./dlkern -s -w
```

You'll notice that I used a -w option as well. This tells dlkern to use **wget** as the means of downloading my new kernel. The default is **ncftpget**. One or both of these programs is likely already on your system or included in your distribution. One quick note here; make sure you change the permissions on the script to executable (**chmod 755 dlkern[**). If you are indeed following the development of a specific driver, one option is to put dlkern into a crontab entry where the download can happen automatically (during the night or whenever makes sense).

So, now we have chosen and downloaded our kernel source distribution. The next step, of course, is to build the kernel. I realize that there are many documents explaining the steps to kernel compilations (other writers in this fine publication have and will, no doubt, cover it in the future). Nevertheless, to truly appreciate the work of our next chef, we should review the process briefly.

With source in hand, it is time to uncompress and extract that source. The usual and preferred location tends to be /usr/src. Kernels direct from any of the kernel.org mirrors tend to extract into a hierarchy called “Linux”. This would mean that your kernel sources lived in /usr/src/linux. However, you may find that /usr/src/linux is actually a symbolic link to where the actual kernel source lives. For instance, if I do an **ls -l** on one of my servers, I get something that looks like this:

```
# cd /usr/src
# ls -l linux
lrwxrwxrwx  1  1  root  11 Nov 14 1999 linux -> linux-2.2.5
```

Notice that it points to a directory called “linux-2.2.5”. If you want to extract directly from the source, start by renaming the link to something else, then extract the kernel. The tree will be called “Linux”. Now, change directory to the linux directory and type **make config**. Even before you type that command, I should tell you that this is the old, one might even say, *classic*, way of doing things. You will get a line-by-line question and answer session designed to help you through the configuration process for your new kernel. If you type **make menuconfig**, you will get a ncurses-based screen that is a bit friendlier, what you might call a GUI for the console set, *non*? The last option is **make xconfig**, which uses an X environment for building.

As I mentioned, this is the question and answer session. You decide what you want in your kernel by answering “Y” for yes, “N” for no and “M” for module. At any point in the process, you can ask for help if you don't know what a specific thing does. The kernel make process is pretty good at leading and helping out through this. If it tells you that you should include it, then do so.

Once this is done, you can actually go back and make some modifications if you want to double-check. In the /usr/src/linux directory, you will find a file called .config. Using **pico**, **vi** or whatever editor makes you happy, you can still make changes. Listing 1 is a sample of the .config file from a recent build.

Listing 1. Sample config File

The next step is to do a **make depend** (which handles all the dependencies and creates your Makefiles), followed by a **make clean**. If this is your first build, you will see messages claiming to be cleaning up and deleting files that aren't really there. Now, it's time to actually compile your kernel. You do this by executing **make bzImage**. This next step may take a while, depending on how fast your computer is, so consider sitting down with a nice Beaujolais while you wait. When all is done, you'll find your new kernel in the directory /usr/src/linux/arch/i386/boot. It is, as you might expect, called bzImage since that is what we told the system to make, *non*? Now, you want to include your new kernel image in LILO's list of bootable kernels. For this example, I built the 2.2.17 kernel directly from source. Once that was done, I used my vi editor to modify my /etc/lilo.conf file. Listing 2 shows what it looked like before I started.

Listing 2. /ect/lilo.conf File

You'll notice that I did not change anything else in this example. My default boot is still **vmlinuz-2.2.14-5.0**, which is the stock kernel that came with my system. The lines that start with **image=/boot/vmlinuz-2.2.17** are the ones I added after the fact. I am now going to have my Linux system reread the configuration file by running this command:

```
/sbin/lilo
```

After LILO has worked its way through your /etc/lilo.conf file, it should come back with something like this:

```
Added linux *
Added linux-2.2.17
```

Here is a quick tip from the kitchen, *mes amis*. From time to time, I have been known to say that too much garlic in a Caesar salad is impossible. I joke, but you understand, *non*? I tell you this so that you will better understand the following statement.

You can never run **/sbin/lilo** too many times, but you can run it too few. If you have made changes to your kernel or your /etc/lilo.conf file and you forget to run **/sbin/lilo**, **your system will not boot**. This is extremely important. If you are

not sure whether you have run LILO, then run it again. Sometimes, I run it two or three times just to make sure. I joke, but only a bit, *non*?

We have only a couple of things left to do. Since you have, no doubt, defined several modules to include with your new kernel, you need to make them as well. You do this with the **make modules** command which you then follow up with **make modules_install**.

There you have it. All in all, it's not that the whole process is so complicated. It's just that there are quite a few things to remember, and forgetting that all-important LILO step can cost you a great deal of aggravation (not to mention that it could put a real dent in your wine cellar).

No matter what, building a kernel, for many (including François), is a daunting experience. This is where **buildkernel** comes into play. Written by William Stearns, this is a wonderful little (big) bourne shell script that pretty much automates the entire process. If you have local kernel source, it will use that, otherwise, buildkernel will download the latest stable kernel for you. You can, if you wish, ask the software to get the latest development version instead. It then extracts it, creates the appropriate links and starts the configuration process by launching **make xconfig**. Once you have decided what you want, your job is pretty much done.

Sometime after building the kernel modules, buildkernel will interrupt you and ask you to verify its changes to /etc/lilo.conf. It does this by putting you into your default editor and allowing you to make changes. Here's what it looks like at this point. This is not the whole file, but simply the changes added by buildkernel:

```
#The following boot section was added by
#buildkernel on Wed Nov 8 13:06:21 EST 2000
#Please feel free to move this section, edit it
#and remove these comments.
image=/boot/bzImage-2.2.17-1
        label=2.2.17-1
        root=/dev/hda1
        read-only
#End of autoinstall
```

Once you are happy with this, exit the editor and let the program continue. It will complete the installation of your new kernel, run LILO and that will be that. No need to worry about whether you forgot some steps. *Oui*, it is true. Despite my assurances that I personally watched as buildkernel ran LILO, I still ran it again.

When you look at the resultant **/etc/lilo.conf** file, notice that the default boot configuration is your old kernel—buildkernel has left that intact. *Merci*. Now, to

book your new kernel, simply shut down your system and type (in the case of my example above) **2.2.17-1** at the LILO prompt.

There are some parameters that you may be wise to pre-set. These are in a control (or config) file called /etc/bkrc, and they allow you to select defaults such as your local mirror download site (BKFTPSITE), the type of image (BKBUILDTYPE), or whether you want to use the stable or development kernels (BKKERNELTOBUILD).

It seems that closing time has arrived very quickly. *Mon Dieu*! Once again, I want to thank you all for visiting my restaurant. François, please, another glass of wine for our friends. As you can see, working with the Linux kernel need not be a frightening experience. With a little help from dedicated open-source chefs around the world, anybody can sample the Linux kernel up close and personal. Even François.

Until next time, your table will always be waiting here at *Chez Marcel*.

Â vôtre santé! Bon appétit!

Resources

Marcel Gagé lives in Mississauga, Ontario. In real life, he is president of Salmar Consulting Inc., a systems integration and network consulting firm. He is also a pilot, writes science fiction and fantasy, and edits TransVersions, a science fiction, fantasy and horror magazine (now an anthology). He loves Linux and all flavors of UNIX and will even admit it in public. In fact, he has just finished writing *Linux System Administration: A User's Guide*, coming soon from Addison-Wesley Longman. He can be reached via e-mail at mggagne@salmar.com. You can discover a lot of other things from his web site at http://www.salmar.com/marcel/.

Archive Index Issue Table of Contents

Advanced search

The 101 Uses of OpenSSH: Part II of II

Mick Bauer

Issue #82, February 2001

Bauer explains the keys to security.

Most people who use SSH never get past its simplest two functions: encrypted remote shells and encrypted file transfers (which is as far as we got last month in this column). That's fine; there's no point in learning features you don't need. But many of you highly self-motivated readers doubtlessly stand to benefit from at least some of SSH's other 99 uses. So let's get down to the *really* cool features of SSH, specifically those of OpenSSH.

Note: throughout this article I'll use “SSH” to refer to Secure Shell generically, i.e., “the Secure Shell system”. Specific Secure Shell processes will be displayed in fixed-width font and in lowercase, e.g., **ssh, sshd,** etc. This is consistent with my other articles: if it looks like something you'd see typed at a console-prompt, it probably is.

Public-Key Cryptography

A complete description of public-key cryptography (or “PK crypto”) simply won't fit in an article that's supposed to be about OpenSSH. If you're completely unfamiliar with PK crypto, I highly recommend the RSA Crypto FAQ (available at http://www.rsasecurity/rsalabs/faq/) or, even better, Bruce Schneier's excellent book *Applied Cryptography*.

For our purposes here, it's enough to say that in a public-key scheme each user has a pair of keys. Your **private key** is used to digitally sign things, and to decrypt things that have been sent to you. Your **public key** is used by your correspondents to verify things that have allegedly been signed by you and to encrypt data that they want only you to be able to decrypt.

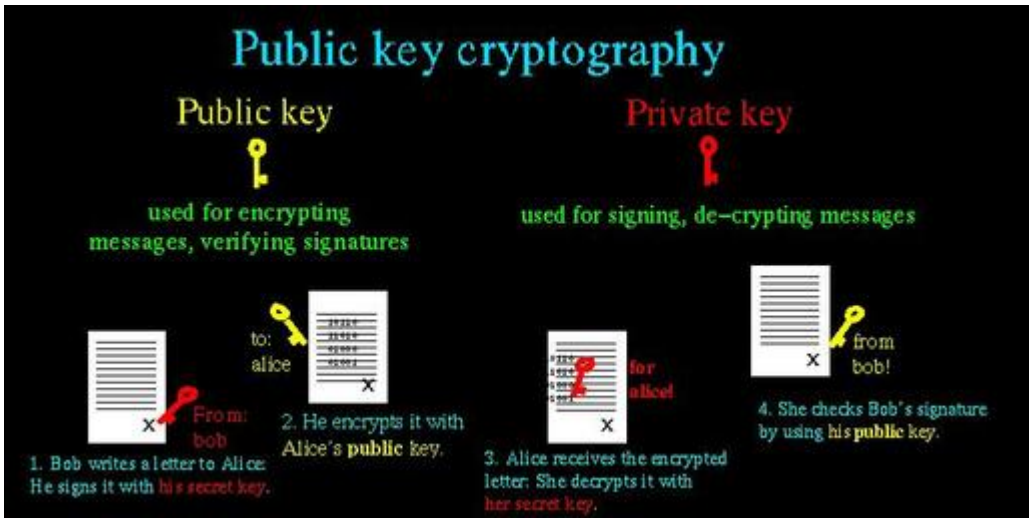


Figure 1. Public-Key Cryptography

Along the bottom of Figure 1 we see how two users' key pairs are used to sign, encrypt, decrypt and verify a message sent from one user to the other. Note that Bob and Alice possess copies of each other's public keys but that each keeps their private key secret.

As we can see, the message's journey includes four different key actions: (1) Bob signs a message using his private key; (2) Bob encrypts it using Alice's public key (NOTE: aside from the fact that Bob has probably kept a copy of the original message, he can *not* decrypt this message—only Alice can); (3) Alice receives the message and decrypts it with her private key; and (4) Alice uses Bob's public key to verify that it was signed using his private key.

Compared to block ciphers such as blowfish and IDEA, in which the same key is used both for encryption and decryption, this may seem convoluted. Unlike block ciphers, for which secure key-exchanges ensure that both parties obtain the key without exposing the key to eavesdropping or other attacks, PK crypto is easier to use securely.

This is because in PK schemes two parties can send encrypted messages to each other, without first exchanging any secret data whatsoever. There is one caveat: public-key algorithms are slower and more CPU-intensive than other classes of cryptographic algorithms such as block ciphers and stream ciphers (e.g., 3DES and RC4, respectively). As it happens however, PK crypto can be used to securely generate keys that can be used in other algorithms.

In practice, therefore, PK crypto is often used for authentication (“are you really you?”) and key-negotiation (“which 3DES keys will we encrypt the rest of this session with?”) but seldom for the bulk encryption of entire sessions (data streams) or files. This is the case with SSL, and it's also the case with SSH.

Advanced SSH Theory: How SSH Uses PK Crypto

Key-negotiation is one of the very first things that happens in any SSH transaction, and the thing that enables a relatively weak form of authentication (username/password combinations) to be used. How the Diffie-Hellman Key Exchange Protocol works is both beyond the scope of this article and complicated (for more information, see the Internet Draft entitled "draft-ietf-secsh-transport-07.txt", available at http://www.ietf.org/). You need only to know that the result of this large-prime-number hoedown is a session key that both parties know but which has not actually traversed the as-yet-unencrypted connection.

This session key is used to encrypt the data fields of all subsequent packets via a "block cipher" agreed upon by both hosts (transparently but based on how each SSH process was compiled and configured). Usually one of the following is used: Triple-DES (3DES), blowfish, or IDEA. Only after session-encryption begins can authentication take place.

But before we dive into RSA/DSA authentication, let's pause for a moment and consider how key-negotiation can be transparent, given that it uses PK crypto and that private keys are usually passphrase-protected. SSH uses two different kinds of key pairs: **host keys** and **user keys**.

A host key is a special key pair that doesn't have a passphrase associated with it. Because it can be used without anybody needing to enter a passphrase first, SSH can negotiate keys and set up encrypted sessions completely transparent to users. Part of the SSH installation process is the generation of a host key (pair). The host key generated at setup time can be used by that host indefinitely, barring root compromise. And Because the host key identifies the host, not individual users, each host needs only one host key. Note that host keys are used by **all** computers that run SSH regardless of whether they run only the SSH client (ssh), SSH dæmon (sshd), or both.

A user key is a key associated with an individual user and is used to authenticate that user to the hosts he or she initiates connections to. Most user keys must be unlocked with the correct passphrase before being used.

User keys provide a more secure authentication mechanism than username/password authentication (even though all authentication occurs over encrypted sessions). For this reason, SSH by default always attempts PK authentication before falling back to username/password.

When you invoke SSH, SSH checks your $HOME/.ssh directory to see if you have a private key (named "id_dsa"). If you do, SSH will prompt you for the key's

passphrase and will then use the private key to create a signature that will send, along with a copy of your public key, to the remote server.

The server will check to see if the public key is an allowed key (i.e., belonging to a legitimate user and therefore present in the applicable **$HOME/.ssh/authorized_keys2** file). If the key is allowed and identical to the server's previously stored copy of it, the server will use it to verify that the signature was created using this key's corresponding private key. If this succeeds, the server will allow the session to proceed, possibly after also performing username/password authentication.

(Note: the above two paragraphs refer to the DSA authentication used in SSH Protocol v.2; RSA authentication is slightly more complicated but, other than using different filenames, is functionally identical from the user's perspective.)

PK authentication is more secure than username/password because a digital signature cannot be reverse-engineered or otherwise manipulated to derive the private key which generated it; neither can a public key. By sending only digital signatures and public keys over the network, we ensure that even if the session key is somehow cracked, an eavesdropper still won't be able to obtain enough information to log on illicitly.

Setting up and Using RSA and DSA Authentication

Okay. You're ready to enter the next level of ssh-geekdom by creating yourself a user key pair. Here's what you do.

First, on your client system (the machine you wish to use as a remote console) you need to run **ssh-keygen**. You've got some choices: among other things, you can specify either RSA or DSA keys; key-length; an arbitrary “comment” field; the name of the key-files to be written; and the passphrase (if any) the private key will be encrypted with.

Now that RSA's patent has expired, choosing the algorithm is somewhat arbitrary. On the other hand, the SSH protocol v.2, which is the version submitted to the IETF for consideration as an Internet Standard, uses DSA keys. If you don't care either way, I recommend DSA. But if for any reason you want RSA then go for it. The **-d** flag sets DSA as the algorithm, otherwise RSA is the default.

Key-length is a more important parameter. Thanks to Adi Shamir's “Twinkle” paper (which describes a theoretical but plausible computer capable of brute-force-cracking RSA/DSA keys of 512 bits or less), I highly recommend you create 1024-bit keys; 768 is okay, but not noticeably faster to use than 1024. However, 2048 is definitely overkill: it isn't significantly more secure, and slows things

down noticeably. The default key-length is 1024, but you can use the **-b** flag followed by a number to specify a different one.

The “comment” field is not used by any ssh process: it's strictly for your own convenience. I usually set it to my e-mail address on the local system. That way, if I encounter the key in **authorized_keys** files on my other systems, I know where it came from. To specify a comment, use the **-C** flag.

The passphrase and file names can, but needn't be provided in the command line (using **-N** and **-f**, respectively). If missing, you'll be prompted for them.

Listing 1. Generating a DSA User-Key Pair

In Listing 1, I'm creating a DSA key pair with a key length of 1024 bits and a comment-string of mbauer@sprecher.enrgi.com. I let **ssh-keygen** prompt me for the file to save the key in. This will be the name of the private key, and the public key will be this name with “.pub” appended to it.

In this example I've accepted the default filename of id_dsa (and therefore id_dsa.pub). I've also let **ssh-keygen** prompt me for the passphrase. The string of asteriks (“**********”) won't actually appear when you enter your passphrase—I inserted those into the example to indicate that I typed a long passphrase that was not echoed back on the screen.

By the way, passphrases are “all or nothing” propositions: your passphrase should either be empty (if you intend to use the new key as a host key or for scripts that use SSH) or should be a long string that includes some combination of upper- and lower case letters, digits and punctuation. This isn't as hard as it may sound; for example, a line from a song with deliberate but unpredictable mispellings can be easy to remember but difficult to guess. The more random the passphrase, the stronger it will be.

That's all that must be done on the client side. All that needs to be done at each remote machine you wish to access from this host is to add the new public key to **$HOME/.ssh/authorized_keys2** (where “$HOME” is the path of your home directory). **authorized_keys2** is a list of public keys (one per very long line) that may be used for login by the user whose home directory **authorized_keys2** resides in.

To add your public key to a remote host you have an account on, simply transfer the file containing your public key (**id_dsa.pub** in the above example) to the remote host and concatenate it to your **authorized_keys2** file. How you get the file there doesn't matter a whole lot. Remember, it's your *public* key, so if it were to be copied by an eavesdropper en route, there would be no need for

concern. But if you want to be paranoid about it, simply do a **scp ./id_dsa.pub remotehostname:/your/homedir**--see last month's column for **scp** instructions. To then add it to **authorized_keys2**, log on to the remote host and enter:

```
cat id_dsa.pub >> .ssh/authorized_keys2
  (assuming you're in your home directory)
```

That's it! Now whenever you log in to that remote host using SSH, the session will look something like what you see in Listing 2.

Listing 2. Logging in Using DSA Keys

Notice that when I invoked **ssh** in the listing, I used the **-2** flag: this instructs SSH to try SSH Protocol v.2 only. By default Protocol v.1 is used, but v.1 only supports RSA keys and we just copied over a DSA key. Note also that the key is referred to by its local path/filename: this is a reminder that when we use RSA or DSA authentication the passphrase we enter is only used *locally* to "unlock" our locally stored private key and is *not* sent over the network in any form.

There's one last thing I should mention about the simple example above. It makes two assumptions about the remote server: (1) that I have the same username as I do locally and (2) that the remote server recognizes SSH Protocol v.2. If the first assumption isn't true I need to use the **-l** flag to specify my username on the remote host. (Alternatively, I can skip **-l** and instead use scp-style username@hostname syntax, e.g., mick@zippy.pinheads.com.)

If Protocol v.2 isn't supported by the remote **sshd** dæmon I'll have to try again without the **-2** flag and let SSH fall back to username/password authentication, unless I've got an RSA key pair whose public key is registered on the remote machine.

To do all this with RSA keys we follow the same steps but with different filenames:

1. Create an RSA user-key pair with **ssh-keygen**, for example:

```
ssh-keygen -b 1024 -C mbauer@homebox.pinheads.com
```

1. On each remote host you wish to connect to, copy your public key onto its own line in the file **authorized_keys** in your **$HOME/.ssh** directory. (The default filenames for RSA keys are identity and identity.pub.)

Again, if you run **ssh** without the **-2** flag, it will try RSA authentication by default.

What happens if you forget your RSA or DSA key's passphrase? How will you get back into the remote machine to change the now-unusable-key's

authorized_keys file? Not to worry: if you attempt RSA or DSA authentication and fail for any reason, SSH will revert to username/password authentication and prompt you for your password on the remote system.

If you wish to disable this “fallback” mechanism and maintain a strict policy of RSA/DSA logins only, change the parameter **PasswordAuthentication** to “no” in **sshd_config** on each remote host running **sshd**. As long as we're talking about the server-side of the equation, note that by default **sshd** allows both RSA and DSA authentication when requested by an **ssh** client process. The **sshd_config** parameters to explicitly allow or disallow these are **RSAAuthentication** and **DSAAthentication**, respectively.

Making Strong Authentication Simpler with ssh-agent

Establishing one or more user keys improves authentication security and harnesses more of SSH's power than username/password authentication. It's also the first step in using SSH in shell scripts. There's just one small obstacle to automating the things we've done with PK crypto: even though the PK crypto-based authentication is transparent, the preliminary key-authorization isn't. How can we safely skip or streamline that process?

There are several ways. One is to create a key with no passphrase, in which case none will be prompted for whenever the key is used. (We'll talk about passphrase-less keys in a moment.) Another way is to use **ssh-agent**.

ssh-agent is, essentially, a private-key-cache in RAM that allows you to use your private key repeatedly after entering its passphrase just once. When you start **ssh-agent** and then load a key into it with **ssh-add**, you are prompted for the key's passphrase, after which the “unlocked” private key is held in memory in such a way that all subsequent invocations **ssh** and **scp** will be able to use the cached, unlocked key without re-prompting for its passphrase.

This might sound insecure, but it isn't. First, only a **ssh-agent** process' owner can use the keys loaded into it. For example, if “root” and “bubba” are both logged in and each have started their own **ssh-agent** processes and loaded their respective private keys into them, they cannot get at each other's cached keys; there is no danger of bubba using root's credentials to run scp or ssh processes.

Second, **ssh-agent** listens only to local **ssh** and **scp** processes; it is not directly accessible from the network. In other words, it is a local service, not a network service per se. There is no danger, therefore, of an outside would-be intruder hijacking or otherwise compromising a remote **ssh-agent** process.

Using **ssh-agent** is fairly straightforward: simply enter **ssh-agent** and execute the commands it prints to the screen. This last bit may sound confusing, and it's certainly non-instinctive: before going to the background, **ssh-agent** prints a brief series of environment-variable declarations appropriate to whichever shell you're using that must be made before you can add any keys. Simply select these commands using your mouse and right click to paste them at a command prompt to execute them (see Listing 3).

Listing 3. Invoking ssh-agent

In Listing 3, I'm one third of the way there: I've started a **ssh-agent** process, and **ssh-agent** has printed out the variables I need to declare using BASH syntax.

All I need to do now is select everything after the first line above and before the last line (as soon as I release the left mouse-button this text will be copied), and right click over the cursor on the last line (which will paste the previously selected text into that spot). I may need to hit the enter key for that last **echo** to be performed, but that echo isn't really necessary anyhow.

Note that the above cut and paste will work in any xterm, but for it to work at a tty (text) console **gpm** will need to be running. If all else fails, you can always type the declarations manually.

Once **ssh-agent** is running and SSH_AUTH_SOCK and SSH_AGENT_PID have been declared and exported, it's time to load your private key. Simply type **ssh-add** followed by a space and the name (with full path) of the private key you wish to load. If you don't specify a file, it will automatically attempt to load $HOME/.ssh/identity. Since that's the default name for an RSA user-private-key, if yours is named something else or if you wish to load a DSA key you'll need to specify its name, including its full path, e.g., **ssh-add /home/mbauer/.ssh/id_dsa**.

You can use **ssh-add** as many times (to load as many keys) as you like. This is useful if you have both an RSA and a DSA key pair and access different remote hosts running different versions of SSH (i.e., some that only support RSA keys and others that accept DSA keys).

Passphrase-Less Keys for Maximum Scriptability

ssh-agent is useful if you run scripts from a logon session or if you need to run **ssh** and/or **scp** repeatedly in a single session. But what about **cron** jobs? Obviously, **cron** can't perform username/password *or* enter a passphrase for PK authentication.

This is the place to use a passphrase-less key pair. Simply run **ssh-keygen** as described above, but instead of entering a passphrase when prompted hit the enter key. You'll probably also want to enter a filename other than "identity" or "id_dsa", unless the key pair is to be the default user key for some sort of special account used for running automated tasks.

To specify a particular key to use in either an **ssh** or **scp** session, use the **-i**. For example, if I'm using **scp** in a **cron** job that copies logfiles, my **scp** line might look like this:

```
scp -i /etc/script_dsa_id /var/log/messages.*
   scriptboy@archive.g33kz.org
```

When the script runs, this line will run without requiring a passphrase: if the passphrase is set to <Enter>, SSH is smart enough to not bother to prompt the user.

But remember, on the remote-host-side I'll need to make sure the key in **/etc/script_dsa_id.pub** has been added to the appropriate **authorized_keys2** file on the remote host, e.g., **/home/scriptboy/.ssh/authorized_keys2**.

CAUTION: always protect *all* private keys! When in doubt, **chmod go-rwx private_key_filename**.

Using SSH to Execute Remote Commands

Now it's time to take a step back from all this PK voodoo to discuss a simple feature of SSH that is especially important for scripting: remote commands. So far we've been using the command **ssh** strictly for remote shell sessions. However, this is merely its default behavior; if we invoke **ssh** with a command line as its last argument(s), SSH will execute that command line rather than a shell on the remote host.

For example, suppose I want to take a quick peek at my remote system's log as seen in Listing 4.

Listing 4. Running cat on a Remote Host

As shown in Listing 4, the host "zippy" will send back the contents of *its* **/var/log/messages** file to my local console. (Note that output has been piped to a *local* **more** process.)

Two caveats are in order here. (1) Running remote commands that require subsequent user interaction is tricky and should be avoided—with the exception of shells, **ssh** works best when "triggering" processes that don't

require user input. Also, (2) all authentication rules still apply: if you would normally be prompted for a password or passphrase, you still will. Therefore, if using SSH from a **cron** job or in other noninteractive contexts, make sure you're either using a passphrase-less key or that the key you are using is first loaded into **ssh-agent**.

Before we leave the topic of SSH in scripts, I would be remiss if I didn't mention “rhosts” and “shosts” authentication. These are mechanisms by which access is automatically granted to users connecting from any host specified in any of the following files: **$HOME/.rhosts, $HOME/.shosts, /etc/hosts.equiv**, and /etc/shosts.equiv.

As you might imagine, rhosts access is incredibly insecure since it relies solely on source-IP addresses and host names, both of which can be spoofed in various ways. Therefore, rhosts authentication is disabled by default. shosts is different: although it appears to behave the same as rhosts, the connecting host's identity is verified via host-key-checking; furthermore, only root on the connecting host may transparently connect via the shost mechanism.

By the way, *combining* rhosts access with RSA or DSA authentication is a very cool thing to do, especially when using passphrase-less keys. See the **sshd(8)** man page for details on using rhosts and shosts with SSH, with or without PK authentication.

TCP Port-Forwarding with SSH: VPN for the Masses!

And now we arrive at the payoff (and the section for which I saved room by sacrificing a more complete discussion of rhosts/shosts): port-forwarding. **ssh** gives us a mechanism for executing remote logins/shells and other commands; **scp** adds file-copying. But what about X? POP3? FTP proper? Fear not, SSH can secure these and most other TCP-based services!

Forwarding X applications back to your remote console is extremely simple. First, on the remote host edit **/etc/ssh/sshd_config** and set “X11Forwarding” to “yes” (in OpenSSH version 2x, the default is “no”). Second, open an **ssh** session using the authentication method of your choice from your local console to the remote host. Third, run whatever X applications you wish. That's it! Needless to say (I hope), X must be running on your local system; if it is, the remote application will send all X output to your local X desktop.

Listing 5. Forwarding an xterm from a Remote Host

After the **xterm &** command is issued, a new xterm window will pop up on the local desktop. I could just have easily (and can still) run Netscape, the GIMP or

anything else my local X server can handle (and which is installed on the remote host, of course).

X11 is the only category of service that SSH is hard-coded to automatically forward. Other services are easily forwarded using the **-L** flag. Consider the session shown in Listing 6.

Listing 6. Using ssh to Forward FTP

The first part of the **ssh** line looks familiar: I'm using SSH Protocol v.2 and logging on with a different username (mbauer) on the remote host (zippy) then locally (mick@homebox). The **-f** flag tells **ssh** to fork itself into the background after starting the command specified by the last argument, in this case **sleep 20**. This means that the **ssh** process will sleep for 20 seconds instead of starting a shell session.

Twenty seconds is plenty of time to start our tunneled FTP session, which is actuated via the magic following the **-L** flag. **-L** defines a “local forward”: a redirection from a local TCP port on our client system to a remote port on the server system. “Local forwards” follow the syntax **local_port_number**: **remote_hostname**: **remote_port_number** where **local_port_number** is an arbitrary port on your local (client) machine, **remote_hostname** is the name or IP address of the server (remote) machine, and **remote_port_number** is the number of the port on the remote machine you wish to forward connections to.

Note that any user may use **ssh** to declare local forwards on high (greater than or equal to 1024) ports, but only root may declare them on privileged ports (less than 1024). Staying with the above example, after **ssh** “goes to sleep” we're returned to our local shell prompt and have 20 seconds to establish an FTP connection. Note that in Listing 6 I'm using **ncftp**: this is because **ncftp** supports the **-p** flag, which allows me to tell it to connect to my local-forward port of 7777. Note also that I give **ncftp** the name of my *local* system rather than the remote host's; **ssh** will take care of routing the connection to the remote host.

After 20 seconds the **ssh** process will try to end, but if an FTP session using the local forward is still active, then **ssh** will return a message to that effect and remain alive until the forwarded connection is closed. There's nothing to stop us from opening a login shell rather than running a remote command (we'll just need to omit the **-f** flag and then use a different virtual console or window to start FTP, etc.). If we do use **-f** with **sleep**, we are not obliged to sleep for exactly 20 seconds—the sleep-interval is unimportant (as long as it leaves enough time to start the forwarded connection).

For that matter, you can run any remote command that will achieve the desired pause, but it makes sense to use **sleep** because that's the sort of thing **sleep** is for. One more tip: if you use a given local forward every time you use ssh, you can declare it in your very own configuration file in your home directory, **$HOME/.ssh/config**. The syntax is similar to that of the **-L** flag:

```
LocalForward 7777 zippy.pinheads.com:21
```

In other words, after the parameter name “LocalForward” you should have a space or tab, the local port number, another space, the remote host's name or IP address, a colon but no space and the remote port number. You can also use this parameter in **/etc/ssh/ssh_config** if you wish it to apply to all **ssh** processes run on the local machine.

Those are some of the advanced uses of SSH and OpenSSH. For now I must bid you adieu and refer you to the man pages for further details and still more features. Go forth and encrypt!

Mick Bauer (mick@visi.com) is security practice lead at the Minneapolis bureau of ENRGI, a network engineering and consulting firm. He's been a Linux devotee since 1995 and an OpenBSD zealot since 1997, taking particular pleasure in getting these cutting-edge operating systems to run on obsolete junk. Mick welcomes questions, comments and greetings.

Archive Index Issue Table of Contents

Advanced search

Linux Enters Router Market

Linley Gwennap

Issue #82, February 2001

Another potential market widens for Linux.

With Internet traffic doubling every six months, networking is one of today's hottest markets. Linux has always played a key role in servers, but now the door is open for Linux to penetrate the heart of the Internet: the high-speed router. These devices, made popular by Cisco Systems, create the Internet by moving data from place to place.

Today, Cisco owns about 80% of the router market. Its products use a proprietary software stack called IOS to handle all of the routing functions. But, as the router market continues to grow rapidly, many new companies are attempting to dethrone Cisco. Rather than designing their products entirely from scratch, many of these vendors use a new model that relies on third-party hardware and software.

The catalyst for the new model is the network processor. This new device burst onto the scene a year ago and has already racked up more than 100 design wins with routers and other networking equipment. It replaces the custom silicon that Cisco and others painstakingly develop for each new product. Intel, Motorola, IBM, AMCC (through its recent purchase of MMC Networks), Lucent and Vitesse have all jumped into the network-processor market, and several start-ups are also developing products.

Internally, network processors are much like standard CPUs, but their architectures are optimized for handling the packet data that passes across the Internet. They bring the advantage of programmability to routers, allowing network equipment vendors to add features more rapidly to their systems, even in the face of quickly evolving standards. Buying standard silicon can eliminate the expensive and time-consuming design cycle for custom chips.

This model greatly reduces the barriers to entry that have helped Cisco repel invaders for the past several years. In this new world, the router market will begin to look like the PC market, with many new vendors jumping in building equipment using off-the-shelf silicon. This competition will reduce Cisco's dominance and provide router customers with more choices and lower prices.

To enable a more PC-like market, however, standard hardware must be joined by standard software. This is where Linux can play a role. Linux would not run on the network processor itself, which executes a fairly limited set of code at high speeds. But every router contains one or more control processors that oversee the system and perform any complicated or unusual tasks. For example, the control processor updates the routing table, an ever-changing map of the Internet, and translates any packets that are not in the standard protocols or formats.

For the control processor, many router vendors use internally developed software, such as Cisco's IOS, or a real-time operating system such as VxWorks. But some are moving toward Linux as a more open environment. For example, MMC recently joined with MontaVista Software to demonstrate an open-router platform running Hard Hat Linux. The platform was based entirely on standard silicon from MMC as well as open-software from MMC and MontaVista. Using these products as a starting point, an OEM could quickly develop and deploy a networking system that competes with Cisco's products. Even with the emergence of off-the-shelf hardware and software, the router market won't become as commoditized as the PC market. PCs are customized by their application software, whereas router vendors must develop all of the software that their equipment will use. Off-the-shelf products can greatly accelerate time-to-market, but OEMs will still add features and customize the product to gain an advantage in a competitive market. Most of this customization will be in the control-processor software. For this application, Linux has the advantages of a robust toolset and source-code availability.

As the networking market expands, more opportunities emerge for Linux. As the Internet grows, much of the new traffic is coming, not from corporate networks, but rather from DSL modems, cable modems and wireless connections. Network processors are starting to play a role in the aggregation points for these connections (known as DSLAMs, cable head-ends and cellular base stations, respectively). Where off-the-shelf hardware leads, open software is likely to follow.

Linux does not have an open path in this market; traditional RTOS vendors are also pushing their products. But, the networking community is certainly familiar with the cost and time-to-market benefits of Linux. Control-processor software can reach millions of lines of code, so software development becomes a critical

issue in networking products. Smashing through $10 billion in annual revenue, the router market presents another sizable opportunity for Linux.

Linley Gwennap (linleyg@linleygroup.com), founder and principal analyst of The Linley Group, is co-author of A Guide to Network Processors (www.linleygroup.com/npu).

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Teaching System Administration with Linux

R. Robert Adams

Carl Erickson

Issue #82, February 2001

A college lab creates new system administrators.

System administration is a vital necessity of any computer system. However, most universities don't teach system administration. So where do people learn how to become system administrators? Basically, they have to learn it on their own. The next logical question is: what enables someone to learn how to do system administration? Our answer: system administration requires a fundamental understanding of how operating systems (OS) and networks operate. However, unlike a traditional computer science operating system or networking course, which teaches low-level (i.e., programming) details, system administration requires only an understanding of theory and fundamentals. For example, you don't have to know about page table layouts to understand how to install and configure swap space.

Here at Grand Valley State University we've developed a course that teaches operating system and networking fundamentals while using system administration as an underlying theme. Our students are information systems majors who would otherwise never learn about the principles of OSes and networking. In the course we cover OS topics such as users, groups, file sharing and processes, along with networking topics such as application layer protocols, the transport layer and network device configuration.

The course has two components: traditional lecture where we teach the concepts and principles of OS and networking, and a lab where the students are able to apply the concepts learned in lecture to a “real-world” environment. Other papers of ours (see Resources) discuss the organization of the course. Our purpose here is to show how Linux is used to support the lab for the course.

The EOS Linux Lab

The Exploratory Operating System (EOS) lab consists of 24 Pentium IIIs with 128MB RAM, 10GB HD, a floppy and a Zip disk. Each machine is running Red Hat 6.2. The lab is a production environment—it serves as the primary account of most CS and IS majors as well as several faculty. Thus, the lab is not a pure research lab. Real people use the lab every day. Because our lab isn't a dedicated research lab, giving root access to a group of 24 students every semester is out of the question. However, the students require superuser access to perform even the most basic system administration duties.

Our solution is to take advantage of the 100MB Zip disk on each machine to provide a dedicated Linux distribution to each student. Each student creates a boot floppy and a root file system on a Zip disk. With this setup, the student can insert both disks and reboot the machine. The student then has a working Linux distribution all to themselves, and one they are able to administer for themselves. In this environment the student can perform the experiments for the lab that day. When they're finished, they simply shut down the machine, remove their floppy and Zip disks and reboot. The system then comes up in the normal EOS lab configuration.

Floppy Boot Disk Creation

Currently, the floppy boot disk kernel is based on the 2.2.13 kernel, and no special kernel source modification is required. However, we do configure the kernel (using **make xconfig**) in two special ways. First, we configure the kernel with SCSI emulation (CONFIG_CHR_DEV_SG and CONFIG_SCSI are set to true). We have IDE Zip disks and run them under SCSI emulation because it seems that the IDE driver doesn't handle large files well.

Our second configuration modification is to disable all access to the hard disk. Remember, we normally run a standard, multiuser Red Hat system in the lab. If we didn't disable access to the hard disk, the student could boot their Zip disk, mount the hard disk and have carte-blanche to make changes (like changing root's password). We disable hard disk access by setting two configuration variables to false, CONFIG_BLK_DEV_IDE and CONFIG_BLK_DEV_HD_IDE.

Other kernel configuration options enable the network device, enable SysV init, etc. Once the kernel is configured, we simply compile it. See the Kernel how-to for more information.

Installing the kernel on a floppy disk is done by creating a new ext2 file system on the floppy (using mke2fs) and copying the kernel to the root of the floppy. The floppy disk also requires a boot block (**cp /boot/boot.b /mnt/floppy**) and a special LILO configuration shown below:

```
boot=/dev/fd0 map=/mnt/floppy/map
install=/mnt/floppy/boot.b
prompt
compact
timeout=50
image=/mnt/floppy/vmlinuz
   label=linux
   root=/dev/sda1
   read-only
```

Our LILO configuration makes the floppy bootable and specifies the /dev/sda1 to be the root disk. Recall that we will be running SCSI emulation, so /dev/sda1 is the Zip disk.

We then run **/sbin/lilo -C /mnt/floppy/lilo.conf** to install the new LILO image.

Zip Root Disk Creation

The root disk is based on Slackware v7.0. We chose Slackware primarily because it gives fine control over what packages are installed, which enables us to easily fit a distribution on a 100MB Zip disk. For our system administration course, we installed the following packages: a, ap and n. Here are the commands:

```
# fdisk /dev/sda        create a single ext2
                            partition that
                            covers the entire
                            Zip disk
# mke2fs /dev/sda1          make a file system on
                            the Zip disk
# mount /dev/sda1 /mnt/zip  mount the Zip disk
# cd /mnt/zip
# tar -zxvf /tmp/slackware/a1/aaa_base.tgz
# sh install/doinst.sh
# rm -rf install
```

Repeat the last three steps for each desired package.

Unfortunately, we had to leave out certain packages for lack of space. Most notably, the d package that provides C/C++ and the k package that provides kernel source were excluded. We would like to remedy this in the future with a larger removable disk. Lineo, a company specializing in embedded Linux, provides an alternative to trimming down a standard Linux distribution in order to get standard UNIX utilities into a small space. BusyBox, an open-source project, combines tiny versions of many common UNIX utilities into a single small executable (see Resources).

Automating the Process

Students create their own boot and root disks in the first lab of the semester. However, they don't have enough knowledge of Linux to do this without a lot of hand-holding. Therefore, we've created a process whereby students can create the boot and root disks by running just a few commands. Specifically, we create

images of a working boot and root disk. Creating the images is done with the following commands:

```
dd if=/dev/fd0  of=floppyimage
dd if=/dev/sda1 of=zipimage
```

Using this approach, students only have to use **dd** to dump the images to the appropriate device with these two commands:

```
dd if=floppyimage of=/dev/fd0
dd if=zipimage    of=/dev/sda1
```

Conclusions

Using Linux to teach system administration has worked very well for us. Using the Zip disk allows each student to get hands-on experience administering their own system without interfering with the "real" system on the hard drive and without interfering with other students. Although the Zip drive is only 100MB, it has proven adequate for creating a working system with all the necessary components.

We strongly feel that our lab would not have been possible without Linux. Because of its open-source nature, we were able to customize the distribution so a complete system would fit on one Zip disk. Furthermore, we were able to customize the kernel to make our production system safe from student tampering.

The only remaining problem with our approach of using a production laboratory environment as a dedicated system administration lab is with the necessary reboots interrupting remote users of the machines. We've addressed this problem by clearly identifying machines that are not subject to these reboots and encouraging remote access users to avoid lab machines with Zip disks.

You may find lecture notes, lab assignments and other software at the course web site (see Resources).

Resources

D. Robert Adams is an assistant professor at GVSU. His research and teaching interests include object-oriented programming, Palm-based computing and programming languages. An associate professor at GVSU,

Carl Erickson is currently on leave of absence working on the software architecture team of XiphNet, Inc. His research and teaching interests are in distributed and object-oriented computing.

Archive Index Issue Table of Contents

Advanced search

Focus on Software

David A. Bandel

Issue #82, February 2001

pam_watch, multiscan, phpopenmonitor and more.

Last month I looked at the Microsoft way and the ugly specrte of a Linux registry. But we have to live with Microsoft (at least for the near term) for other incompatibilities as well. One of those is in printing. The Microsoft way is for every client to know everything about the printer it will be using, instead of the print server, which, under Microsoft, only queues jobs to the printer but doesn't format them. So what happens if you have to change out the printer (or upgrade to a newer one)? Well, you have to track down those several hundred clients and change each one individually. Linux allows you to do this silliness, if you really want to, with raw printers (no print filter usage). But you don't have to. If all Linux boxes that are print servers do the document formatting from generic postscript files, printing becomes a no-brainer. Just tell Word Perfect, et. al., to use a passthrough postscript driver. Windows systems can use the Apple 1200 postscript driver. Now you can print to any printer in the UNIX/Linux world, to fax machines and even send postscript files to friends to read and print (or turn into PDF files). No need to juggle 600+ printer drivers for all the world's printers on every client. Simplicity beats Microsoft because who needs more headaches?

pam_watch: http://frida.fri.utc.sk/~behan/devel/pam_watch/

Do you need to snoop on terminals? Perhaps show someone from afar how to accomplish a particular task? This utility will allow two users to watch and use the same terminal at one time, even continents apart. When used as a login session, pam_watch creates two pipes, one for input and one for output, that someone (usually root) can attach to. The only downside is that it won't work on ptys (used in X sessions and ssh sessions) or sessions spawned from the terminal. Requires: libpam, libdl, glibc.

multiscan: http://sourceforge.net/projects/multiscan/

I think we all know that nmap is good. But it's not fast, and it's a little heavy. If you need a quick scan of your own network to see which ports are open on which systems, and you need it yesterday, multiscan will tell you. I watched it rip through a Class C private network in no time. Granted, unreachable hosts slowed it down a lot, but reachable hosts showed all open ports at a pace of two hosts per second. That's fast. Requires: glibc.

phpopenmonitor: http://www.edomex.net/phpopenmonitor/

Do you need to be able to check a number of systems for open ports (running services) often and quickly? This utility won't take the place of nmap—you can't search for open ports. But if you list ports you want to be sure are open (or closed) and enter those next to the host name, you can see at a glance if all is well. It's quick and easy to set up and autorefreshes every five minutes (feel free to change that). Requires: web server, php4, web browser (capable of color output).

SQL-Ledger: http://www.sql-ledger.com/

After looking at this particular application, all I can say is Wow! Someone was reading my mind (now that's a scary thought). Take a PostgreSQL server, a little Perl, mix in a web server and all the ingredients for a good accounting program, and you have: SQL-Ledger. Okay, so I haven't checked to see if it's compliant with GAAP (Generally Accepted Accounting Procedures), and it's been years since my last accounting class, but this is good. It still lacks a few details, like POS (point-of-sale), but they're on the to-do list. With a program like this, who needs Quick Books? Requires: PostgreSQL, web server, Perl and Perl modules: DBD-Pg, DBI.

CCC: http://www.noguska.com/ccc/

CCC is not a general accounting program (although it could be used as one with some modifications); it does an excellent job of quantifying work for a computer maintenance shop. You can track jobs, technicians and systems. You can use the information to bill clients. If you need a simple job tracking/ invoicing system, this could be what you need. Requires: MySQL, web server with PHP and MySQL support, web browser.

tvguide: www.cherrynebula.net/tvguide.html

Sound and video aren't usually my thing. Okay, so when I'm working on something I might listen to a little Pink Floyd—“Comfortably Numb” is good music to concentrate on a problem with. But I do like to keep an eye on the news, or whatever, something to do while a build (or two) is in progress. With

tvguide, I can grab what's on the site quickly, grep it (or just read it) and tune in something interesting on my TV card. Might be a waste of good cycles, but they're often idle anyway. At least I don't miss as many football games. Requires: Perl.

pkgbuild: http://www.linuxsupportline.com/~davin/

If you need to build RPM packages, this little GUI tool can help you. While you still have to know how to build one, this utility makes a good aid. I will warn you that some particular incantations within the spec files are rejected. But if you start with a good template (that pkgbuild likes) you can go easily from there. This tool will not make you an RPM wizard, but it will force you to relook at how you structure your spec files (if you consider that a plus). Requires: libm, libSM, libICE, libXext, libX11, glibc.

sniffer: http://stev.org/

This utility will show you a great deal about what's happening on on your network in terms of the bandwidth you're consuming per minute, etc. You can see stats of TCP, UDP, ARP and other packets. You can switch between different displays. While only root can run it (unless you permit users to open raw sockets), it is a very handy tool. You can also see MAC addresses and a good guess of the Ethernet card brand. Requires: libpthread, libncurses, glibc.

Until next month.

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

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

A Survey of Embedded Linux Packages

Rick Lehrbaum

Issue #82, February 2001

Brief looks at the various embedded Linux projects and applications.

Because Linux is openly and freely available in source form, a great many variations and configurations of Linux distributions have evolved in order to match the needs of a broad range of markets and applications. There are implementations ranging from handheld devices including PDAs and cell phones, to Internet appliances, thin clients, firewalls, telephony infrastructure equipment and even supercomputer clusters. There are small-footprint versions and real-time enhanced versions. And despite the origins of Linux as a PC operating system, there are now ports to numerous non-X86 CPUs, with and without memory management units, including PowerPC, ARM, MIPS, 68K and even microcontrollers. And there's more coming, all the time!

One of the most interesting characteristics about embedded Linux, is the abundance of choice. Even for a given architecture, such as PowerPC, there tends to be an abundance of available alternatives. So, how do you decide which distribution to use? That depends.

First, it's important to realize that all Linux distributions are, more or less, variations on the same theme. That is, they tend to be collections of the same basic components, including the Linux kernel (of course), drivers, command shells (command processors), GUI and windowing frameworks, utilities and libraries.

So, what makes them different? The distinctions between distributions frequently center around which of the many hundreds of Linux utilities are included, what modules or utilities (both open-source and proprietary) are added, kernel patches and modifications, and how the installation, configuration, maintenance and upgrade process is managed. How easy is it, for example, to build a small-footprint Linux system that closely matches your application's unique requirements?

Then there's the question of proprietary open-source software. This might be an important philosophical issue for you or your company. You (or your company) may want to restrict your Linux-related activities to using exclusively open-source software so there are no licensing restrictions or royalty requirements, and so that you can supply source code to your customers. Or, you might prefer a more pragmatic approach, using (and sometimes licensing) whatever software components best match your application's needs—whether proprietary or open-source.

Either way, whether you decide to use pure open source, or to mix open and proprietary software components, using Linux in embedded applications is unlikely to be totally “free” of costs. You'll either invest resources to create your own implementation, money for tools and licensed components, or pay for outside services and support.

That said, there are certainly a wide variety of excellent open-source tools and utilities that you can download free of charge; and, there are also excellent proprietary tools and utilities that you would need to license or purchase. Then, too, bear in mind that the companies that offer “commercial” embedded Linux distributions generally possess a high level of expertise and have well-trained staff ready and waiting to assist you in your project. Paying money to one of the “commercial” embedded Linux suppliers can have many advantages, including development tools, useful utilities—and, of course, support. The proprietary embedded Linux suppliers' companies are actively investing in developing tools and services to differentiate their Linux offerings from the pack, and (or) to advance their standing as a prospective partner for companies building embedded applications that will have Linux inside. In many cases, they are also contributing to the overall pool of open-source software.

Taken as a whole, there are a great many options—both open-source and proprietary—that fall into these general areas:

- Tools to automate and simplify the process of generating a Linux configuration that is tuned to a specific embedded system's requirements.
- Graphical user interfaces (GUIs), windowing environments and browsers that vary in size, appearance, features and capabilities.
- Drivers and utilities that support the specific needs of a diverse range of applications including telephony equipment, multimedia devices, mobile computing, wireless capabilities and more.
- Linux kernel enhancements and add-ons that support the “hard”, “firm”, or “soft” real-time performance requirements of such applications as streaming media, IP telephony, machine control, etc.

What follows, are brief descriptions of many of the currently available commercial and noncommercial sources for embedded Linux solutions. Bear in mind that new projects and products come into being on an almost daily basis. For this reason, LinuxDevices maintains a set of continually updated on-line embedded Linux Quick Reference Guides that are available for free public access.

"Commercial" Embedded and Real-Time Linux Distributions

Coollogic: Coollinux—Coollinux AE (Appliance Edition) combines the power of embedded Linux and Java technology to deliver an operating system for the next generation of Internet appliances. http://www.coollogic.com/.

Coventive: XLinux—fully featured embedded Linux kernel that can be configured to as little as 143KB for information appliances and embedded devices. Support: 586, 686, MediaGX, STPC, StrongARM, SH3/SH4, PA-RISC, ARM-7 and more. http://www.coventive.com/.

Esfia: RedBlue Linux—an embedded Linux distribution for wireless communication solutions, derived from the prerelease Linux 2.4 kernel. It has a typical kernel, footprint is 400K bytes and it supports processors with or without an MMU. http://www.esfia.com/.

FSMLabs: RTLinux—FSMLabs provides a real-time Linux distribution based on RTLinux technology in conjunction with a Linux kernel and associated software. MiniRTL is a small-footprint (fits on one floppy) implementation for resource-constrained embedded applications. http://www.fsmlabs.com/.

KYZO: PizzaBox Linux—a Linux- and Samba-based file, print and CD server designed to run from 6MB of Flash ROM on a 486 (or higher) CPU. http://www.jrcs.co.uk/.

Lineo: Embedix—an embedded Linux-based software solution that is engineered specifically for the unique speed, memory and storage requirements of embedded devices. Supports a wide range of CPUs with and without MMUs, including X86, PowerPC, ARM, MIPS and more. Includes support for small-footprint, real-time (based on a choice of RTAI or RTLinux) and high-availability solutions. http://www.lineo.com/.

LynuxWorks: BlueCat—a distribution of open-source Linux, enhanced to meet the requirements of embedded developers, engineered to allow configuration to accurately match the requirements of embedded development from small devices to large-scale multi-CPU systems, and high-availability applications. Supports a wide range of CPUs including X86, PowerPC, ARM, MIPS and more. http://www.lynuxworks.com/.

Mizi: Linuette—a Mobile Linux OS that provides components, development tools and a specialized kernel for mobile devices, designed to satisfy the small size requirements of the SmartPhone market. The target hardware environment is an 18MHz ARM7 processor, 240x120 pixel LCD display, touch screen and serial interface. The OS requires just 2MB of DRAM and 4MB of Flash memory. http://www.mizi.com/en/.

MontaVista: Hard Hat Linux—the MontaVista Software Hard Hat Linux Cross Development Kit targets a broad array of embedded CPU architecture boards and system-level platforms for Internet appliances, portable devices, networking equipment, telephony interfaces, or other embedded and pervasive applications. http://www.mvista.com/.

PalmPalm: Tynux—an embedded Linux solution optimized for Internet appliances including MP3 players, video players, Internet TVs, PDA/cell phones, Internet phones, video conferencing equipment, video phones, etc. http://www.palmpalm.com/.

REDSonic: RedIce-Linux—a real-time Linux distribution that provides several enhancements to real-time performance including RTAI as well as enhancements to the Linux kernel's scheduler and preemption algorithms. http://www.redsonic.com/.

TimeSys: Linux/RT—a real-time Linux distribution that offers multiple means to improve real-time performance, including a resource kernel, RTAI, Linux kernel scheduler, preemption and quality-of-service enhancements. http://www.timesys.com/.

"Noncommercial" Embedded and Real-Time Linux Implementations

ART Linux—a real-time extension to Linux (developed by Youichi Ishiwata) which was inspired by RTLinux but which, according to its developer, "offers certain advantages." http://www.etl.go.jp/etl/robotics/Projects/ART-Linux/.

Embedded Debian Project—the goal of the Embedded Debian Project is to make Debian GNU/Linux a natural choice for embedded Linux. http://www.emdebian.org/.

ETLinux—a complete Linux distribution designed to run on small industrial computers, especially PC/104 modules. http://www.etlinux.org/.

KURT—a real-time Linux implementation that allows scheduling of events with a resolution of tens of microseconds. Based at the University of Kansas. http://www.ittc.ukans.edu/kurt/.

Linux Router Project—a "networking-centric micro-distribution" of Linux that makes it easy to build/maintain routers, access servers, thin servers, thin clients, network appliances and embedded systems. LRP can fit on a single floppy. http://www.linuxrouter.org/.

Linux/RK—a "resource kernel" enhancement to Linux based on a loadable kernel module that provides timely, guaranteed and enforced access to system resources for applications. Based at Carnegie Mellon University. www.cs.cmu.edu/~rajkumar/linux-rk.html.

LOAF—"Linux on a Floppy" distribution that runs on 386s and is an implementation of Linux consisting of the kernel and a bunch of free utilities. LOAF supports various network protocols including the lynx browser, ftp, Telnet and ssh. http://loaf.ecks.org/.

Linux-SRT—an extension to the Linux kernel that improves the performance of "soft real-time" applications such as multimedia but not suitable for the critical timing requirements of hard real-time apps, like controlling space shuttles or nuclear reactors. http://www.uk.research.att.com/~dmi/linux-srt/.

Linux-VR—the goal of this project is to support Linux on NEC VR series devices, most of which were originally designed to run Windows CE-based handheld computers. http://www.linux-vr.org/.

uClinux—a derivative of the Linux 2.0 kernel intended for microcontrollers without Memory Management Units (MMUs). Supports a growing list of processors including Motorola DragonBall (M68EZ328), M68328, M68EN322, ColdFire, QUICC; ARM7TDMI; MC68EN302; Axis ETRAX; Intel i960; PRISMA; Atari 68K; and more all the time! http://www.uclinux.org/.

uLinux (a.k.a. muLinux)—a "full-configured, minimalistic, almost complete, application-centric tiny distribution" of Linux, made in Italy, that fits on a single floppy. http://sunsite.auc.dk/mulinux/.

PeeWeeLinux—a small Linux distribution aimed at embedded devices. The distribution attempts to make the configuration and installation of a Linux operating system on an embedded platform as painless as possible. http://www.peeweelinux.com/.

QLinux—a Linux kernel implementation that provides Quality of Service (QoS) guarantees for soft real-time Linux performance in applications such as multimedia, data collection, etc. Based at the University of Massachusetts. http://www.cs.umass.edu/~lass/software/qlinux/.

RED-Linux—a real-time version of Linux that implements short kernel blocking time, quick task response time, a modularized and runtime replaceable CPU scheduler and a general scheduling framework. Based at the University of California, Irvine. http://linux.ece.uci.edu/RED-Linux/.

RTAI—a comprehensive Real-Time Application Interface usable both for uniprocessors (UPs) and symmetric multiprocessors (SMPs) that allows the use of Linux in many hard real-time applications. As an option, RTAI's "LXRT" allows the control of real-time tasks, using all of RTAI's hard real-time system calls from within Linux memory-protected user space, resulting in soft real time combined with fine-grained task scheduling. The RTAI project is based at the Dipartimento di Ingegneria Aerospaziale Politecnico di Milano (DIAPM). AtomicRTAI is a small-footprint implementation for resource-constrained embedded applications. http://www.rtai.org/.

RTLinux—a hard real-time mini operating system that runs Linux as its lowest priority execution thread. The Linux thread is made completely preemptive so that real-time threads and interrupt handlers are never delayed by non-real-time operations. The latest version of RTLinux supports user-level real-time programming. MiniRTL is a small-footprint (fits on one floppy) implementation for resource-constrained embedded applications. http://www.rtlinux.com/.

ThinLinux—a Linux distribution for embedded and dedicated applications, designed to be run on minimal Intel and PC-compatible hardware. http://www.thinlinux.org/.

GUI and Windowing Environments for Embedded Devices (Commercial and NonCommercial)

Century Software: Microwindows PDA Operating Environment—a fully functional screen-top, web browser, terminal emulator, pop-up keyboard and handwriting recognition system as the basis for development and execution of Linux-based applications for the iPAQ and other PDAs. http://embedded.centurysoftware.com/.

Lineo: Embedix UI—a low footprint, HTML-based user-interface solution for embedded devices. Provides an easy way to create an aesthetically pleasing interface while adding functionality in a wide variety of devices; it is well suited for controlled-content embedded devices including webpads, information appliances, kiosks, screen phones, set-top boxes, point-of-sale terminals, home entertainment devices, industrial automation, etc. http://www.lineo.com/.

Compaq: The Open Handheld Program—Compaq established the Open Handheld Program in order to stimulate innovation and research on handheld devices (e.g., PDAs). The effort was seeded with the results of the earlier

Compaq “Itsy” pocket computer project. A developmental Linux port for the Compaq iPAQ is currently available for download. http://www.handhelds.org/.

The Microwindows Project—an open-source project aimed at bringing the features of modern graphical windowing environments to smaller devices and platforms. Microwindows applications are built and tested on the Linux desktop, as well as cross-compiled for the target device. http://www.microwindows.org/.

The TinyX Project—a small footprint X Windows system server implementation for embedded systems. It was developed by Keith Packard of the XFree86 Core Team, sponsored by SuSE. The goal was to create something that would work well in a small memory footprint and, importantly, be robust in near out-of-memory situations. Typical X Windows system servers based on TinyX can fit on less than 1MB in X86 CPUs. http://www.xfree86.org/

Transvirtual Technology: PocketLinux PDA Framework—a Linux-based PDA software environment that integrates Kaffe (a “clean room” implementation of Java) with embedded Linux and also provides built-in support for XML. PocketLinux also includes an integrated frame-buffer graphics library that eliminates the need for a resource-hungry X Window System. http://www.pocketlinux.org/.

Trolltech: Qt/Embedded—Qt/Embedded features an API that is identical to the existing Qt/X11 and Qt/Windows products. However, Qt/Embedded is not based on X11 and, therefore, it has substantially lower memory requirements than X11. By picking and choosing features, the memory demands can be tuned from 800KB to 3MB in ROM (Intel X86). http://www.trolltech.com/.

Trolltech: Qt Palmtop Environment—a complete PDA Linux software package, including a Window System, Window Manager, Application Launcher, Input Methods (virtual keyboard, etc.), GUI toolkit and collection of useful applications, all written using the standard Qt API—the same API found on Qt/X11 and Qt/Windows. http://www.trolltech.com/.

The ViewML Project—a freely available, open-source web browser targeted specifically at the embedded Linux platform. Currently, ViewML along with its interface requires 2.1MB of RAM, with a disk image of only 760K. http://www.viewml.org/.

Rick Lehrbaum (rick@linuxdevices.com) created the LinuxDevices.com “embedded Linux portal”, which recently became part of the ZDNet Linux Resource Center. Rick has worked in the field of embedded systems since 1979. He cofounded Ampro Computers, founded the PC/104 Consortium and was instrumental in launching the Embedded Linux Consortium.

Archive Index Issue Table of Contents

Advanced search

Finality

Stan Kelly-Bootle

Issue #82, February 2001

The last, last word.

My dear faithful reader-fans who have survived my diverse ramblings since I first rambled way back in *Computer Weekly* circa 1956 will know that one of my most piquant threads has been of the psephological persuasion. As you know, psephology is the formal study of elections, apparently trivial but dripping with deep, dark paradoxes. There are endless papers in the MAA and AMS MathLit searching for the “perfect” electoral methodology. When voters are asked to record their “preferences”, it seems that the “normal” logical/algebraic relational laws break down. Thus, the transitive “A prefers B and B prefers C implies A prefers C” often fails when the total votes are counted (see Referecnes).

Or re-re-counted, to refer to the current (as I write in November 2000) Florida Presidential fiasco that will decide “the lesser of two weasels” as Jay Leno stated. My Brazilian correspondent, Rainer Brockerhoff, is alarmed/amused by the fact that the leading “wired” nation (aka “Fellow 'Mericans”) requires “hand counts” while the billions in Rio and São Paulo are “cast” electronically.

Apart from the quirks (popular vs. electoral college) of choosing the leader of the free-world, we are subjected to and influenced by statistically dubious polls daily.

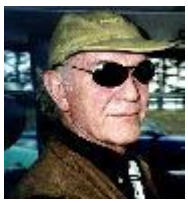
I helped install/program IBM for George Gallup in the 1960s. The in-joke was the over loaded quiz: “Given the rampant inflation and unemployment, would you vote again for that Commie bastard Harold Wilson?”

Evans Data and similar “opinion” gatherers are more subtle, yet we need to be cautious. What if 39.78% (note the spurious accuracy) of developers say they are “scale 6 of 10” interested in “looking at C# when it's defined”?

Finally, as my last “Last Word”, I bid a fond Abschied.

I'll continue to promote the Linux agenda in my other columns.

Resources



Stan Kelly-Bootle (skb@atdial.net) has commented on the unchanging DP scene in many columns (“More than the effin'Parthenon”--M eilir Page-Jones) and books, including The Computer Contradictionary (MIT Press) and *UNIX Complete* (Sybex). Stan writes monthly at http://www.sarcheck.com/ and http://www.unixreview.com/. Visit his home page at http://www.feniks.com/skb/.

Archive Index Issue Table of Contents

Advanced search

A Talk with Tim O'Reilly

Doc Searls

Issue #82, February 2001

“The only game that's interesting to Tim is to come up with a vision, or a series of visions, and watch them unfold—and see exactly how he can realize each by hacking the system.” --Eric S. Raymond

At the first Linux World Expo after the turn of the millennium (January 31-February 3, 2000), I had a brief talk with Tim O'Reilly that I recorded and then lost. Recently, when that tape reappeared, I gave it a listen and found that it was as relevant as ever—perhaps even more so—because Tim is one of those rare businesspeople who not only takes the longest and broadest possible view but acts constantly on Yogi Berra's most impossible advice: “When you see a fork in the road, take it.” Looking back, Tim seems to have taken every fork along the way and sometimes several at once. Yet, there also seems to be something consistently broad and encompassing about every one of those decisions.

In retrospect, the vision Tim shared in that talk seems exceptionally knowing and prescient, especially since I was mostly interested in just one question: *Why didn't he sell out*? Even though publishing companies are not held in especially high regard by venture capitalists (who tend to value businesses with potential valuations in the billions over ones with actual revenues in the mere millions), O'Reilly has a long track record of growth and success. More significantly, it is one of the most influential companies in the software world.

Yet O'Reilly expressed almost no corporate vanity—at least not compared to other companies in its communities that were either already public or headed that way. Relatively speaking, O'Reilly & Associates never seems to care about selling itself, beyond whatever it takes to publicize its various products and services. It has always been, in the oddly discreet vernacular of finance, private. And Tim has told me more than once that he intended to keep it that way.

I wondered why. For freshly public Linux companies, Y2K was a heady time, as all of them were still surfing a tsunami of fashion investing. Red Hat, Cobalt Networks, Andover and VA Linux Systems had all gone public in late 1999 to enormous run-ups in valuation. At the time I spoke to Tim, VA Linux had just bought Andover.net (now the OSDN division of VA Linux) for more than $1 billion in stock. Less than two months earlier, VA went public to the largest first-day run-up in history, closing at $239 after trading as high as $320. Red Hat had gone public only four months earlier and peaked at close to $270 around the time of VA's debut. When I spoke to Tim, VA was trading in the $120s and Red Hat at around $100. Even though those numbers were way off their early peaks, both companies were very, very hot, and Linux along with them.

Today the Linux investment tsunami is flat as a pond. VA Linux shares now (on November 20, 2000) sell for around $13 and Red Hat's for around $10. Meanwhile O'Reilly & Associates continues to truck along in its merry, myriad ways, looking far the wiser for not selling at any price.

Here's the conversation with Tim. The date was February 1, 2000.

Doc: One of my pals here just suggested that more money was spent on booths at this show than all the Linux companies in the world combined made in revenues last year. That make sense to you?

Tim: [Laughing] No, but they have a job to do and a lot of money to spend.

Doc: Linux is hot right now. You're a major player in the category. It seems to me that you could easily sell out if you wanted to. Yet you haven't. Why?

Tim: This is really the second big explosion that we've been an interesting part of. The first was the Web. The second was Linux and open source. There were others before the Web, actually. And there will be others after Linux. We've been around long enough to know that every trend is more like a wave than a line that goes up toward the sky. One wave builds on another. When you identify with any one of them, you get a short term advantage out of it, but then you get locked into that stage in history. We could have been a UNIX company, or a PC company, or a groupware company, or an on-line company, or an Internet company, or a web company or, now, a Linux company. Trust me: another wave is going to come after this one. Linux will still be here, but it won't be the hot thing it is right now. Nor should it be.

Doc: You just don't want to get locked in, then.

Tim: Right. It's just never interested me very much. We've just been around too long. We're good at what we do, and we have a role to play; we'll just keep on playing it.

Doc: I used to have a partner who said, “Don't say no, say how much.” That's another way of saying everything has a price. If that's so, and you're still saying no—and companies like Andover, which had a small fraction of O'Reilly's revenues last year, are selling for upward of a billion dollars—then you're telling me that you value O'Reilly & Associates somewhere in the billions.

Tim: [Laughing] I suppose so.

Doc: Okay, so you like to take the long view. Give me some background on that.

Tim: Like a lot of people, I got started doing programming. And, like a lot of people who do programming, I saw a need for manuals. So, the company got started in 1978 as a technical writing consultancy specializing in manuals. In the early eighties, open systems were getting hot, and we started licensing our manuals to various UNIX vendors. Gradually our consulting business gave way to a publishing business.

Doc: But you've been more than a publishing business. You've been highly involved with, rather than just selling to, a market or a category.

Tim: We do like being part of the industry and what makes the industry change over time. I think we probably have a better sense of where things are going and the pulse of technology, than most companies do, mostly because we've been around a long time, and we cover a lot of topics.

Doc: You're not tied down to any one category, or title or whatever.

Tim: No, we're not. Whatever programmers and other technologists need, we're there publishing a book about it.

Doc: Or running a conference.

Tim: Or starting something like GNN, which we sold to AOL.

Doc: I had forgotten about that.

Tim: There are a few things I'd like to forget.

Doc: I'd like to talk a bit about the issue of intellectual property, which I know you care pretty deeply about. Jeff Bezos did us the favor of putting the issue of software patents on the front burner when he obtained the patent for one-click

shopping and promptly sued Barnes & Noble for infringing on it. You've been very vocal about the whole matter. I'd like to hear what IP (intellectual property) means to you.

Tim: I think of intellectual property as topsoil. Hard to make, easy to waste. It's a rich substrate of ideas, techniques and opportunities that anyone can use. That's a rich soil on which you can build business opportunity. We should all be mindful of that and contribute to it. But there are some business models, particularly business models that create hype and fund themselves out of the stock market rather than providing real value to customers, that don't contribute. These models leech nutrients out of the system. If you can convince customers that they should not have to pay for high-quality technical information, for example, nobody will produce it. What some of these companies have done is exploit some short-term conditions that allow them to take a whole bunch of money out of the pockets of potential shareholders while, at the same time, destroying the conditions that made their success possible. So, it's like hey, there's all this stuff that we could get for next to nothing. We'll pretend that we have a viable business model by selling it cheaper than the cost of production. This is just like what happens with resource exploitation in the physical world. Hey, air and water are free! Somebody once asked Ted Turner what he would build a business on if he were a young man, and he said “Water.” And he's buying up watersheds all over the place. He knows there is stuff that has real value that's undervalued at the current time.

Doc: And you think this is the case with software IP?

Tim: I just see a lot of danger in business models that are predicated on some resource that was not created or enriched by the people who exploit it. Again, this is one of the imperatives of the Linux community. So far it's doing better than the Internet community did. I joined the Internet Society Board back in '95 because I saw this happening with the Internet. I had hoped they could model themselves on the Sierra Club. I thought the environmental movement was a great model of people saying, “I have some kind of right in this common good. I can speak for it even if I don't own it.” There is this common heritage represented by the Internet culture, by Internet software, by open-source software, that is being exploited in the same way that natural resources have been exploited. Users need to have some voice that says “don't do that to us.”

Doc: So we'd say to Amazon, “Don't sue Barnes & Noble over your patent on a natural resource. Beat them some other way.”

Tim: Exactly. You can compete but don't compete in a way that's destructive to the substrate that you're building on. Again, I think that the Linux and Open

Source communities have done a much better job than the Internet community did five years ago. Linus and various Linux companies have done a better job of giving back to the community that depends on the environment the community is creating. They are much more conscious of what we hold in common. But I don't think it's as far-reaching yet as it needs to be. We need people thinking about this over a very long period of time. I don't know if you've seen the book, *The Clock of the Long Now*, by Stewart Brand. I was at a lecture he gave with Brian Eno in which Eno talked about speaking to this woman who lived in a wonderful house in a lousy urban setting. When he asked her how she liked living here, she said it was terrific. But he realized she was talking about "the small here", not "the large here". He said he realized that a lot of our problems in society come from our definition of what we're going to care about, how local it is, and we often miss the larger picture. Or worse, we obscure it by painting a small one on top of it.

Doc: You see this as relevant to our current infatuation with the stock market rather than the larger market that surrounds it.

Tim: Yes. We have a lot of businesses optimized for the short term and for a narrow set of interests. They don't spend a lot of time asking, "What will the big benefits be for the culture as a whole?"

Doc: How far back does open-source consciousness go?

Tim: We've talked a lot about open-source software, but the IBM PC was just as big a revolution in its time—and still is. It was open-source hardware. That's what made it cloneable. Its openness freed up a tremendous wave of innovation. But what happened on top of that was the building of a new proprietary empire that was even more oppressive than IBM was in the hardware days. I fear that we're actually going to get there again in spite of Linux. I've been saying for years that just because one layer of software is free doesn't mean that the next layer will be free. We can expect to see closed software being built on free and open-source software, where the new applications will be fundamentally closed. You can look at Amazon as an application. Or Mapquest. Or E-Trade. These guys are not touched by open-source licenses. The GPL does not matter if you don't distribute software. E-Toys won't give away their e-commerce software because they don't have to—which may be fine. I have no problem with that. But I do note that these developments are not part of the ecosystem. They don't give back to the common soil that we all grow from.

Doc: Do you prefer the GPL to other licenses?

Tim: The one thing I don't like is the compulsive aspect of it. I mean, people can choose not to use it, but it also keeps some people from using it. But my point is that it's most critical to establish the culture in which the companies that build on the open platform realize it's in their interest to keep it going. Microsoft has more to gain from feeding the Open Source community than by competing with it.

Doc: I think it's inevitable that Microsoft will support open source in a variety of ways over time.

Tim: Possibly. But there really are a lot of choke points here. To me, the fundamental insight that Microsoft had in the software world was that there are choke points, and if you can hold those passes, you can control the markets that depend on them. And believe me, many people in the e-commerce and the Web world are thinking the same way. And some of them are getting a pretty good stranglehold on some of those choke points, and they will inevitably have others dancing to their tune. I am very concerned that we are going to see in this new layer a number of players who try to dominate by controlling choke points. They may not be as dominant as Microsoft overall, but they will be just as dominant in their own categories.

Doc: What about the big media players—such as AOL/Time Warner?

Tim: I'm not too concerned about the big media conglomerates. Well, we do need to be concerned about the ways we get broadband. The sources matter, in terms of how fast broadband is deployed, what kind of access they provide, and who controls it. But the fact is, the active producer/passive consumer model doesn't work very well anymore. We're all too connected. Those of us on the leading edge really do think of markets as conversations.

Doc: We see supply and demand as symmetrical.

Tim: Right. And we like technology that works the same way. It should be of interest to readers of *Linux Journal* that when we did our web site, we deployed the first NT-based web server. Why did we, a UNIX company, do that? Well, we saw at the time that the Web was breaking down the fundamental model. The vast majority of people using browsers were using them on Windows, and there were no Windows servers. So I used to joke that the Web was becoming the world's largest read-only groupware system. That was in fact the model. What we tried to do was keep alive the vision of the Web as a peer-to-peer medium. In fact, we did.

Doc: How did it work out?

Tim: Not very well as a business, but it did get Microsoft into the web server market. For all the bad stuff one can say about it, the people using that platform can publish to the Web. Prior to that, we were hearing all this stuff about push technology from people who were trying to make the Web look like television instead of an interactive model.

Doc: They were trying to weld the Web to the rusty back end of television's history.

Tim: That's right. And this choice of ours to deploy NT as a web server was an example of trying to think in a bigger way. If we said, “Our primary allegiance is to Linux, or UNIX, and we wouldn't be caught dead using NT,” we wouldn't have done anybody any favors. We said, “Hey, the users are using Windows, and they're getting a bum deal here.” We need to give them the same tools as the culture that we want to build. And that's a key concept here. Why did we publish *Windows 98 in a Nutshell*? Because we want to teach people using that platform how to think in the way that the people who are over on UNIX think. I want them to be empowered and show them that they're the bosses. If you look at that book, you'll see it teaches, as much as it can, the UNIX mindset, on this crippled platform.

Doc: Do you bring a similar message to the Linux world from what lies outside?

Tim: A lot of what I try to get across to the Linux community is the importance of the Web. Take “the network is the computer” as the mantra. You've got to be a player there. What I worry about is that so much of the focus is on the last war, which was over operating systems. I kind of feel like the real question is how are we going to play in the world being built on top of those operating systems, regardless of what they are. I look at the web-based open-source projects as what we need to think about more, get involved in more. Apache. SendMail. QMail. Even something like MySQL. Anything web-related is really important. The big question should be, what does the computer of the future look like? I think it all looks like one big computer. So, how are we building the tools to participate in that? For example, what we're doing with Collab.net is figuring out and training people how to work in a collaborative way. So much of the Linux conversation has been about how the roots of the movement have been in the GNU project and free software. I'd like to talk to people about its roots in Usenet. Its roots in the way people shared information. If we think the conversation is about a particular style of software licensing, and that Microsoft ought to put out its software under open-source licenses, and that “we” somehow win if “you” ignorant mortals follow what the priesthood hands down without question, we've all lost. At the end of the day, it's about a culture of communication and innovation that comes from many points and not down from on high. In some sense, we have a chance to build a new world. I see too

many people jumping ship from this new world to go back and work in the old one—or in the part of the old one that's building closed stuff on the open new world we're all building together. So, it kind of goes back to the beginning of this conversation when we were talking about the money. If you don't think long term, you play into the hands of the old order.

Doc: But there is a lot of financial incentive to think and behave short term.

Tim: That's right. And those of us who take advantage of those opportunities need to keep the long term in mind, to keep the common interests in mind. This is a world we're building together here.

Doc: And there's still money to be made there, too.

Tim: Exactly.

After I turned the tape recorder off, I suggested to Tim that he take a look at Jabber, the new open-source instant messaging system. By August, Tim was talking up Jabber at his Open Source conference; and not long after that he joined the Jabber.com board of directors. (I was, and still am, on the advisory board.) Meanwhile Tim's patent reform conversation with Jeff Bezos has morphed into Bounty Quest, a site that involves several communities in the same patent reform effort.

The man just keeps taking those forks.

Doc Searls is senior editor of *Linux Journal* and a coauthor of *The Cluetrain Manifesto*.

Archive Index Issue Table of Contents

Advanced search

Heavy Gear II for Linux

Neil Doane

Issue #82, February 2001

It's a step in the right direction if the direction you want to go is away from the same old boring first-person shooters.

- Manufacturer: Loki Entertainment Software
- E-Mail: sales@lokigames.com/
- URL: www.lokigames.com
- Price: $29.95 US
- Reviewer: J. Neil Doane

Set in the distant future (did I say distant? the 62nd century!) you are the squad commander of an elite assault force of huge humanoid battlefield vehicles called “Gears”. Using carefully planned tactics and sophisticated futuristic weaponry, you lead your squadron through an engaging story line directed against your people's arch nemesis, Earth.

It's a step in the right direction if the direction you want to go is away from the same old boring first-person shooters. *Heavy Gear II* is addictive, with a nice sci-

fi story line, good multiplayer capability, enough single-player modes to keep you interested for days and game play that allows extremely fine control of your in-game combat experience but retains a strong arcade feeling. It features stunning graphics with picturesque landscapes (and spacescapes) and excellent accompanying audio. *Heavy Gear II* isn't perfect; extended game play will begin to expose interesting quirks that tend to become irritating at times, and there are some technical issues that could probably have been handled better (and some that apparently have yet to be resolved). Overall, however, this game's fun factor seems to win out in the end, and the game has little difficulty consuming one's evening hours and free time.

The Story

Developed around the Dream Pod 9 game system of the same name, *HG2* is Activision's follow-up to *Heavy Gear*, and its story line picks back up about two decades after the end of the the first *HG*. To summarize, an interpolar war on your home planet, Terra Nova, (a former Earth colony but now independent) has come to a halt in the face of an explosive and cataclysmic event that appears to herald the return of invasion forces from Terra Nova's former arch-enemy: Earth. The governments of both poles of your planet have agreed to cease hostilities toward each other in order to confront this age-old enemy. As their first step toward the defense of your world they have formed of an elite fighting unit, composed of the best Gear pilots on the planet. You, of course, being the cream of the crop, white-hot ball of Gear-piloting fighting prowess that you are, have been chosen as the team's squad commander. Your mission, basically, is to take your team of combat specialists behind enemy lines to discover what attack plans Earth has in mind for your world and determine their tactical strengths and weaknesses.

What does this all mean to you as a player? Although the story line is relatively simple, it tends to grow on you as it progresses and, unlike many first-person shooter story lines, seems to add quite a bit to the overall feel of the game play. Moreover, the various dialogs and briefings direct your strategy toward different goals as the story line progresses. This gives you a nice change from the seemingly endless begin/fight/kill/end cycles of so many other first-person games. Based on where you are in the story, you'll have to use different tactics to succeed.

Overview

Heavy Gear II has been plagued since its release by comparisons to its competition, *MechWarrior 3*. Though based on similar themes, these games are actually quite different. Gears are physically different from *MechWarrior*'s robots, and the game play style of the two games varies accordingly. With a

Mech, you are a walking battleship, a massive hulking behemoth, with more firepower than a division of M1 A1 Abrams tanks, and could probably find camouflage in a futuristic skyline—as a building. Gears, on the other hand, while still large and extremely powerful, are typically a fraction of the size, the largest being only about 15 feet tall. With a Gear, one must use tactics and wits to a much greater degree than with its larger cousins. Gears are small enough to be sneaky and to use stealth as a valid tactic. In fact, their smaller stature also means, unlike their 20-story counterparts, they cannot stand in battle and just take the abuse given. Therefore, hiding, setting up ambushes and the like are widely used and often mandatory since you'll be sorely outnumbered from time to time.

Game Play

Heavy Gear II isn't simple. This isn't something you can install from your CD and immediately jump into action. *HG2* seems nearly flight simulator-like in complexity when one first sits down to it. In fact, to become thoroughly acquainted with all the controls and modes of game play requires a full seven-course meal of rather involved training missions and should probably include the convenient placement of the included keyboard-shortcut command placard just to keep track of the myriad commands used in the game. The payoff, however, is extraordinary; the level of control a player has in *HG2* is amazing. For instance, even general movement consists of several axes of control, as one must coordinate the separate movements of the upper torso of your Gear to work in concert with the lower portion of your body. In turn, your lower body has several modes of operation, including jump jets, walking, kneeling, crawling and even, in Gearspeak, an SDS or “Secondary Movement System”, which deploys what amount to all-terrain rollerblades from the soles of your Gear's feet to allow you to, quite literally, skate into battle (or away from it) at high speed. Radar modes allow one to scan actively for nearby enemies, increasing the likelihood that they will be able to detect your Gear's presence. Or, you can enhance your Gear's stealth by switching to a passive-only mode. Your Gear is equipped with maneuvering thrusters for control in the weightlessness of space, and the act of accurately space-maneuvering a Gear in all its possible axes of motion while in a combat situation seems as difficult as neurosurgery the first few times at bat.

Figure 1. *HG2*'s Gear Lab Boasts More Customization Options Than You Can Shake a Vibroaxe at.

The amount of customization possible in the game is unparalleled. Since you're fighting for two wealthy governments, your budget isn't really limited; you can have anything you want and as much of it as you can carry. The list of possible Gear models is seemingly unending (by my count about 70 primary models) and divided into four main categories: Light, Medium, Heavy (of course) and Assault. There are typically several base configurations for each primary model, and the Gear lab has the functionality to create and save your own custom configurations as often as you like. As for guns 'n ammo to attach to your Gear, again, the list is truly huge; just the *names* of weapons in the vast array at your disposal take a few minutes to read and a detailed study to understand fully. To create some modicum of balance in the game, a system of “threat levels” has been implemented. Missions have a maximum total threat level that the entire team can have, as well as the maximum total threat level for any single Gear in the team, and you are forced to configure the weapons and equipment of each Gear on the mission so that neither of these limits are exceeded. Added to the mix is a complex system of “Perks” and “Flaws” that can be assigned to alter the threat profile of each Gear (Perks increase the threat score, Flaws decrease it) and make it truly unique on the battlefield. If you're into this sort of highly detailed customization, you won't be disappointed.

Figure 2. *Heavy Gear* Missiles

So, does all of this preplanning and detailed control pay off in superior game play? Well, kind of: it's enough fun to keep you up all night. However, it also has some problems. Activision redesigned the entire game engine for *HG2* (the so-called “Dark Side” engine) and, though it's very nice at times, it seems less robust and has poorer effects than some of its non-Linux-ported competition. The scenery is completely indestructible, allowing no cratering or deformation of any kind except for the exact targets that *HG2*'s programmers thought you should be blowing up (which are typically only the handful of final goal targets, not just any ol' tree or rock along the route). There is also some curious clipping issues with the engine that can cause your Gear to, at times, slide right though mountains, rocks and even walls as though they don't exist. The AI is basically pathetic; although you have a great deal of control of your teammates through the relatively simple team-control commands, these guys are often as brain-dead as hockey pucks. On several occasions, squadmates become stuck in various landscape features, and I once had to destroy an entire base's defenses just to get to a single enemy Gear who was stuck against a wall, trying to go through it to get to me. The landscapes, although a bit dark, are quite spectacular at times, but the clipping distance is too short and, without the fogging effects activated, you can often clearly see the edge of the world dropping off on the horizon or appearing abruptly in the distance as you walk along. Again, these issues, while unsightly, aren't really enough to ruin the fun of the game, but they seem to leave a bad aftertaste at times.

There are several modes of game play available in *Heavy Gear II*. The campaign mode is very cool, the missions are short enough to keep you glued to the action, and the story line is compelling and unfolds in briefings and creative cut scenes that use the game engine to draw the action. Also available is a “Historical” mission section, which lets you relive and refight some of the greatest battles in Terra Novan history. For those of you who simply want basic

combat, and want it right now, there is an “Instant Action” mode of play available that does just what it should; it puts you in the thick of your choice of a wide variety of highly customizable combat situations. There are several missions in the single-player training mode (which this reviewer highly recommends) and, last but not least, the multiplayer mode. *HG2*'s multiplayer mode includes several types of game play: a one-on-one duel-type game; the traditional multiplayer “death match” free-for-all; a “steal the beacon” game where players accumulate points for the length of time they are able to hold onto the “beacon” before someone blows them to bits; a “strategic” variant where players form teams and attempt to destroy their opponents' bases; and, lastly, the classic “capture the flag”. Unfortunately, there are no cooperative multiplayer campaign-style modes in *HG2*. I think this would have added a great deal of fun to the game. However, I've heard that the historical missions were originally intended to be cooperative, but a scarcity of in-game resources forced the game designers to abandon the idea and make them single-player instead.

Figure 3. Shooting It up in Instant Action on the Ice Arena

Linux Issues

Heavy Gear II was originally written in the Direct3D API (from that company in Redmond). It was the first Direct3D game to be ported to OpenGL for Linux and, as such, is a terrific accomplishment by Loki. For my money, they did a great job. Unfortunately, there are issues with the game that are, by and large, not really Loki's fault. This game has many of the same problems in every platform. On my G400-based VA Linux box, things worked fine, though I did experience some of the interesting clipping issues mentioned earlier. However, the closed-source drivers on my NVidia GeForce GTS-based Debian system went belly-up every time I went from 3-D mode back to the 2-D menus and

debriefings, which made it impossible to use the single-player campaign modes at all on that machine. Loki's support gave me some interesting suggestions and patches to try to alleviate this problem, but none were successful in the end. I've since heard that this is a known issue with the NVidia cards and this particular game.

Summary

Heavy Gear II is a great game, and it's highly addictive once you get past the steep initial learning curve. The story isn't just a few cut scenes used to break up the action; it's fun to watch unfold and holds your interest. The overall feel of the game is fast-paced and fun, nearly arcadelike at times; however, the game play problems are hard to ignore. I can't help but feel as though this game was picked from the vine a bit before it had ripened fully. That said, I'd certainly recommend giving *Heavy Gear II* a try, especially if, like me, you really enjoy simulator-style games with some strategy and tactics required. However, if you want a simple and easy-to-learn first-person shooter, you should probably pass on *HG2*.

The Good/The Bad

J. Neil Doane (cainei@valinux.com) is a professional services engineer with VA Linux Systems and an Indiana escapee. Between prolonged spasms of rabid geekness, random hardware scavenging and video gaming, he is a pilot, a guitarist and a very poor snowboarder.

Archive Index Issue Table of Contents

Advanced search

easyLinux v2.2 and easySamba v1.0

Joseph Cheek

Issue #82, February 2001

EasyLinux and easySamba, developed by easy Information Technology and resold by IGEL@USA are intended to be an easy introduction to Linux.

- Manufacturer: Easy Information Technology
- E-mail: info@eit.de, info@igelusa.com
- URL: http://www/eit/de http://www.easylinux.com/
- Price: easyLinux v2.2 $29.95 US (full version [reviewed here, 2 CDs]) or $49.95 US (deluxe version [not reviewed here, 5 CDs])
- Price: easySamba v1.0 $49.95
- Reviewer: Joseph Cheek

EasyLinux and easySamba, developed by easy Information Technology and resold by IGEL@USA (http://www.igelusa.com/), are intended to be an easy introduction to Linux.

The package I received had a boxed version of easySamba and two CDR's of easyLinux, presumably the free-download version. It also included product

brochures on IGEL's Linux software products (easyLinux and easySamba), hardware products (Linux thin clients) and consulting and reseller programs.

The easyLinux setup program is graphical with 11 steps required to install the software. After choosing my language, keyboard and mouse type, I was asked for my experience level with Linux: am I a novice, an experienced user or a Linux professional? The choice determined which configurations it would let me choose on the next screen. The professional choice provides the greatest flexibility, allowing my system to be a stand alone PC, an Internet-connected PC, a LAN client or a server.

At this point I could also choose whether I was installing on a laptop (it wouldn't let me select a laptop installation if I wanted a server) and how I wanted my GUI to appear—like Windows, like traditional UNIX (Motif), like newer UNIX (CDE) or like a Macintosh (Platinum). I chose Windows but noticed that I got a KDE 1 interface no matter which choice I made—only the widgets were changed.

On to partitioning: this step is always a difficult one for new users to grasp. The screen showed me all existing partitions on both of my hard drives and allowed me to choose which ones to use for easyLinux, along with whether to format them and where to mount them. Also shown were the existing NFS mounts I had in my **/etc/fstab** directory from my previous installation. These mounts were carried over into my new installation, a nice touch.

Had I wanted to create new partitions, it gave me options to run both its graphical fdisk tool or plain vanilla text-based fdisk. It didn't allow me to install into an existing Windows partition (as do Red Hat's, Slackware's and others partitionless installs) so I chose an existing ext2partition and Linux swap partition.

Next I was asked to choose my time zone and set up my Ethernet card. After dutifully entering in this information, the file copy started. During the file copy, which lasted about ten minutes, a little penguin strolled around the screen giving advice about easyLinux and telling me what it could do. This was cute.

After the file copy I was asked how to install LILO. I chose to install it in the Master Boot Record of **/dev/hda** and was told it needed to reboot. Time to see what easyLinux looks like!

When the system rebooted I was presented with a nice KDM login screen. I logged in as root (the setup program told me this was the proper procedure, since no user accounts had been created yet) and was given the opportunity to set a password. An 800x600, 16-bit KDE desktop greeted me with over a dozen icons and a help screen. “Step 1”, it explained, “Please start **eXs** to adjust the

graphic card and monitor." Beyond this was information instructing me to configure Ethernet cards and load software (didn't I just do these during the install?), add a normal user account, configure additional hardware and create a rescue disk. Hmm, seven additional steps, two of which are redundant, and then I will have a functioning easyLinux system.

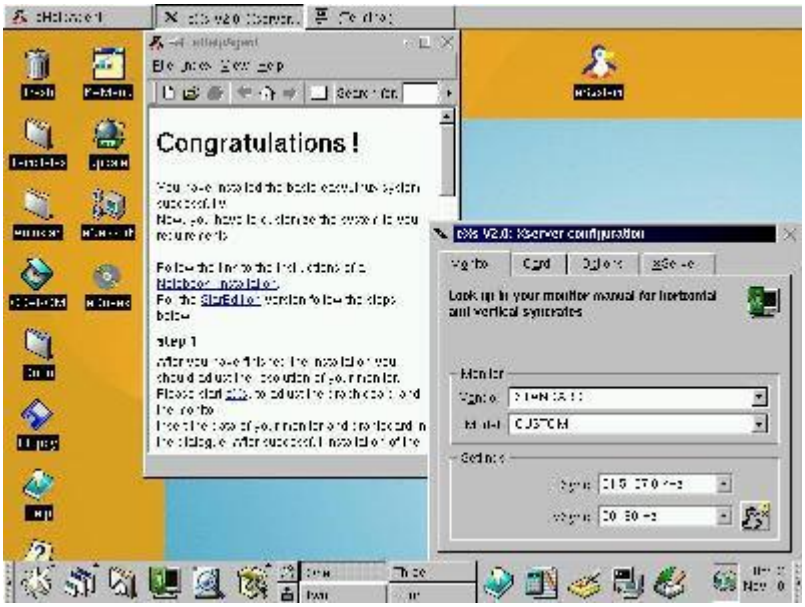


Figure 1. After successfully loading easyLinux, I was greeted with this desktop and told to run eXs, the X configuration program. Here is the on-line help with eXs showing. The desktop is KDE 1.1.2

eXs couldn't get my display to run at 1024x768x32, nor could it get 800x600 for my Trident 3dimage985 video card nor my CTX PR700 monitor (although both were listed in the configuration program). I'm not sure whether this was because it wanted to guess refresh rates (it wouldn't let me choose any beyond **BEST** or **SAFEST**) or because it was using XFree86 3.3.5, but I elected to stay with the working frame buffer X server to which it defaulted.

A quick perusal of eEtherPro, the included Ethernet configuration program, showed my install settings were intact, but no driver was installed for my 3C905B network card. **eAdapterDetect** to the rescue! It found the card and installed the driver right off the bat. Now eEtherPro had the adapter and could start the network drivers.

The third step, install software, is really necessary because the only standard application software I found loaded were two text editors and two MPEG video players. **eProfile**, the application loader, gave me a list of hundreds of applications that I could load from the included CD set. Without a "select all" option, I saw I would have to check each of the hundreds of applications manually to get a full system. I instead checked only a few games, Netscape and sound and video support, and waited for it to install the 78 packages it

calculated I would need (from dependencies and such). After logging out and logging back in, the new programs showed up in the menu.

It should be mentioned here that each time the easyLinux CD is needed, a dialog pops up on the screen stating “Please insert easyLinux CD 1 (or 2).” The CD automounts, using the BSD automounter, **amd.** It works rather well, except that eXs asked for the CD six times before it recognized I already had it in the drive. I ended up ejecting it and reinserting it, and it mounted fine. Sure beats **mount -t iso9660 /dev/cdrom /mnt/cdrom** and **umount /dev/cdrom; eject**.

Adding a user, with **kuser**, was a simple task. Enter full name, user id, home directory, password twice and *voilà!* Configuring a printer brought up Netscape (which I luckily had just installed via **eProfile**) but instead of getting a printer control panel, I got Netscape search results for “localhost631”. The CUPS administration service wasn't started; a quick **/etc/rc.d/init.d/cups start** later, and I was back in business.

Well, sort of. The easyLinux help link to adding a printer wasn't working, but as I figured it out manually I realized my Canon BJC 4400 wasn't supported. The printing software was CUPS, and being familiar with LPRng myself I didn't have the expertise to work around it. Oh well.

Step six, hardware setup with **eHardware**, let me install a sound driver (OSS found my Sound Blaster 16, ALSA didn't). The OSS autoprobe worked well. It would have allowed me to set up a joystick, a TV tuner card and ISA PNP devices had I had any on this system (I didn't).

I chose to skip the last step, create a rescue disk, because I wanted to get to know my new system better. Under the hood, I saw that this was an RPM-based glibc 2.1.3 system with KDE 1.1.2, XFree86 3.3.5 and gcc 2.95.2. Kernel 2.2.16, Netscape 4.74, CUPS 1.1b5 and ALSA 0.5.8 were included. The server install, with the games and multimedia utilities I installed, used 656MB on my hard drive. The system didn't appear to be based on Red Hat or derivatives, Caldera or SuSE, or at least it was disguised enough that I couldn't tell.

The desktop, once I got playing around with it, seemed no different than any other KDE 1.1.2 desktop. No tuning was done to the desktop itself, other than to add a background image with their logo and a few desktop icons. The biggest value adds I saw with easyLinux were the configuration tools.

Beyond the installer and the configuration tools already described, easyLinux bundles a disk wizard (mount new partitions), an FTP setup wizard, an inetd wizard (start and stop inetd services), an ISDN gateway wizard, a printer wizard (which would allow me to install LPRng, but I couldn't get it to work), a RAID

wizard (build a RAID 1 mirror or Linear mode partition), a graphical logfile viewer, a graphical kernel building utility, a registry editor and a list of active services.

All in all, almost two dozen wizards or graphical configuration utilities were included, all integrated with the KDE desktop or configuration manager.

In my mind the question, “If I were a new user, would I want to use easyLinux?” lingered. The second question, “If I were a new user, would I find value in the administration utilities easyLinux has bundled?” I could answer affirmatively. But the larger question: “Would I use this distro?”—I'm not sure I would. Here's why:

- There are a *lot* of bugs. Some are minor annoyances (such as the numerous spelling errors), some are pesky, but not insurmountable (such as the X server not letting me work at 1024x768) and some are pretty bad (easyLinux wouldn't install at all on my laptop, giving me L 80 80 80 80 80 errors when I booted with the CD). Instead of listing all of them here, I'll just say I found dozens in the several hours I played with easyLinux. For version 2.2 of their software, I had expected better.
- The dialogs in the graphical configuration tools are nice but not intuitive. The colors weren't consistent, the buttons weren't placed in the same location, the help icons were scattered all over the dialogs and so on. Some didn't work at all or needed a lot of tweaking to get working. It got very confusing.
- The on-line documentation has pretty good coverage of the different topics but not much depth. Some of the help files had links to files that didn't exist, but generally the help was hard to follow.

This distribution has a great concept, but in order to execute well it could use some more polish. My advice to eIT: stop kitchen-sinking. Remove some of the functionality of your distribution and spend more time making what you leave in work very well for the market you target. Having a single version of a distro work for an Internet desktop, a stand-alone PC, a LAN client and a network server, with customized GUI clients for each little bit of configuration, is an enormous amount of work. Find out which markets you wish to target and focus your distribution a little more on those.

Now to try easySamba. easySamba comes on one CD with a 70-page user manual. Twenty pages of the user manual describe how to set up and configure easySamba, while the other 50 describe Samba, the SMB protocol behind Samba and Windows networking and list security issues with Samba and Windows 95 and Windows for Workgroups (along with how to fix them).

easySamba claims it runs on easyLinux, SuSE, Red Hat and compatible systems. I tried to load it on Redmond Linux, based on Caldera, and even though it detected my system as being Caldera-compatible, it didn't load correctly. I was using kernel 2.4.0-test10, glibc 2.1.95 and Samba 2.0.7, and it overwrote key packages with older versions of the same package. Even after doing so, it would not work correctly.

On my easyLinux system, easySamba installed, well, easily. After inserting the CD into my CD-ROM drive I clicked on the setup script, clicked the forward button and the finish button and was presented with the Samba Wizard.

A few questions were asked: did I want a PDC (Primary Domain Controller), a workstation or a workgroup computer? What did I want for a domain name, and a description for this computer? Did I want to use WINS (the Windows Internet Naming Service)? After entering this and clicking Finish, I was presented with a screen showing the Samba service installed but not active. I activated it and tried to browse from another computer.

Unfortunately, the other computer was the one that easySamba had failed to load on. Perhaps this was the reason I could not see the easySamba server from this computer. I could see it from itself using the **smbclient -L** command, so I was fairly sure the easySamba installation worked well.

At least, as a PDC it worked well. As a workstation it wanted to know the name of the PDC and a Samba PDC wouldn't work. It had to be a real Windows PDC, I gathered. A workgroup computer installation worked well, but not the workstation installation.

The Samba administration program **sambaadmin** was added to the KDE menu after a successful installation. This program allows administration of Samba users, workstation accounts (on a PDC), and file and printer shares. It also shows computers connected to that host.

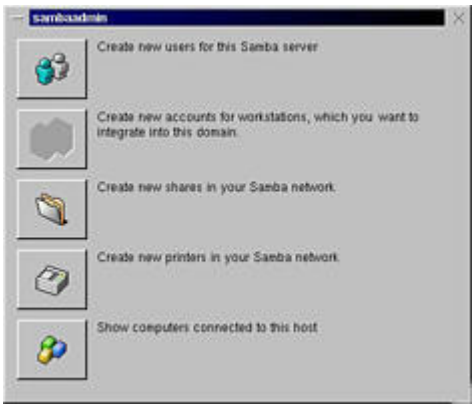


Figure 2. easySamba's GUI configuration tool, samba admin, lets you administer samba user and workstation accounts and file and printer shares, as well as seeing which computers are

attached to your Samba server. The workstation option is grayed out because I chose to do a workgroup computer install, so no workstation accounts were needed.

New file shares are created through **kfm**, the file manager, similar to the way they are done in Windows. Right click on a directory, choose samba share and a dialog pops up asking for a name, a comment, the path (with a default filled in) and which users are allowed to access it. These new shares are automatically accessible, without requiring a manual restart of the samba service.

I created three file shares this way and attempted to edit them via sambaadmin. For some reason only two of them showed up in the GUI dialog, although **smbclient -L** showed all three. Strange.

Parsing through the **/etc/smb.conf** file, I noticed that password encryption and UNIX password synchronization were enabled. No smbpasswd file was defined or populated, which meant that if I had wanted to give my UNIX users Samba access rights I would have to do it by hand. This is not a big deal unless there are many UNIX users to add; I suppose someone in this situation would rather have done this via a script than a GUI anyway.

Security with a network service is always important to me. A guest user, samba, was configured in the **smb.conf** file but not defined in the **smbpasswd** file. After creating a samba user I tried to access a share but was denied due to bad name/password pair. I grepped the logs to find the one easySamba used—**/var/log/syslog**—but couldn't get any clues as to what was wrong. I created a blank password for the samba user. Still no go. I made up a password for this user. *Voilà*, it worked! Guest accounts therefore must have a password. To me, a guest account is one without a password, but I guess that isn't the way everyone thinks.

I decided to forego heavy-duty security and usability testing. The admin interface is fairly nice, and the bugs are relatively minor. easySamba is a little easier to use than Samba's native GUI configuration tool, SWAT, but the only significant value add I see in easySamba is the ability to add file shares via a right-click in kfm. Still, that feature is pretty cool in my book.

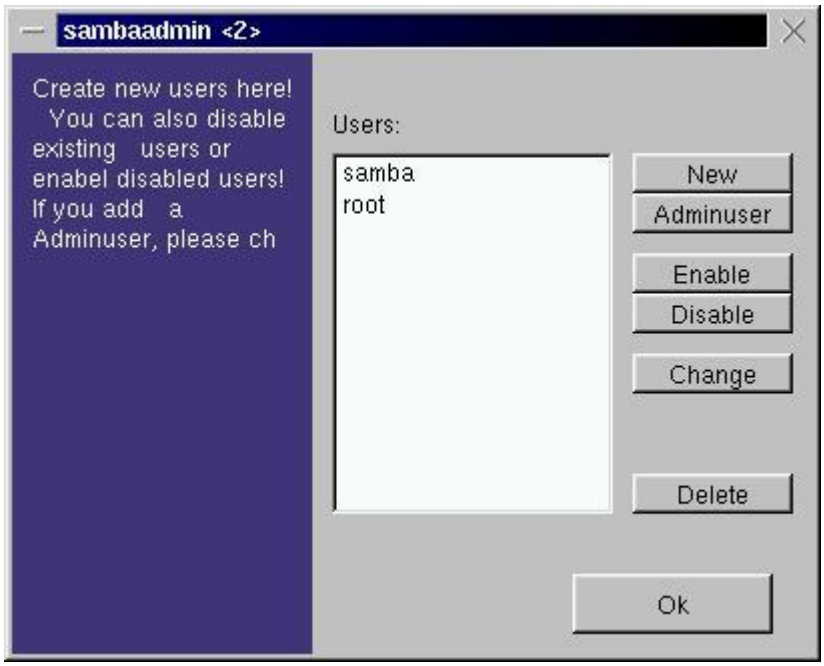


Figure 3. When administering user accounts with easySamba, you can create normal users, administrative users, enable and disable user accounts, delete users and change user passwords.

If I were a small network administrator and wanted to set up a Samba server, would I consider using easySamba? Yes, I would. It generally works as advertised, and it's easy to use and load. Further documentation is missing how to troubleshoot easySamba would be a great addition—Why doesn't the guest account work? Why can't I join an easySamba workstation to an easySamba domain? Why don't all of my shares show up in the configuration tool? How do I check the logs? What should I look for in the logs? Why doesn't the send message button work when I am connected from my own computer? and so on.

And, of course, it should prompt the user before overwriting newer packages with older versions.

The Good and the Bad

Joseph Cheek is a senior Linux consultant with the Professional Services Group in Linuxcare, Inc.'s, Seattle office. He spends most of his free time playing with his wife and two daughters and working on Redmond Linux. He can be reached at joseph@linuxcare.com.

Linux and the New Internet Computer

Billy Ball

Issue #82, February 2001

The New Internet Computer Company may become known for producing and selling the lowest-cost Internet and network capable computer on the market today—the NIC, or New Internet Computer.

- Manufacturer: The New Internet Computer Company
- E-mail: info@thinknic.com
- URL: http://www.thinknic.com/
- Price: $199 US
- Reviewer: Bill Ball

Larry Ellison is well known as one of the first persons to coin the term “Network Computer”, a much-heralded promise of low-cost network appliances and terminals for corporate America. His latest spin-off venture, The New Internet Computer Company, may become known for fulfilling that promise by actually producing and selling the lowest-cost Internet and network capable computer on the market today—the NIC, or New Internet Computer.

At $199 for a main box, speakers, keyboard and mouse, the NIC provides a surprising host of features. A color-matched $129 15-inch monitor is optional. Perhaps the best feature of all is that the NIC runs Linux! All the software you need to start surfing the Web is included on a bootable CD-ROM that sports a Linux 2.2.15 kernel, Netscape Navigator, and connection software for dial-up ISPs, DSL, cable modems or an internal network. This means that you can get to work right away.

As if the price point on this piece of hardware wasn't low enough, you can also use free ISP service from NetZero with your NIC. You're not locked into using the offered service, though, so if you want to use your existing ISP, your company's LAN, or your at-home DSL or cable modem, go ahead.

Linux is finding its way into the computer industry in an ever-increasing variety of platforms, ranging from “Big Iron” servers the size of refrigerators to embedded devices and web servers that fit inside a box of matches. You'll find that the NIC provides a simple, inexpensive way to browse the World Wide Web, use electronic mail and read Usenet news.

You must order your NIC on-line at http://www.thinknic.com/. You'll receive e-mail confirmation of your order, and you can track the shipping through NIC's web site. You'll then receive a box containing the NIC, a keyboard, mouse, mouse pad, amplified speakers, NIC software on CD-ROM, power cords, phone cable, warranty information and a short, ten-page user guide. You'll also get free Internet service from NetZero. This means no confusing service agreements, obfuscated rebates or other insidious deals aimed at enticing consumers to the Dark Side's network.

Initial Impressions

The NIC is a single-board computer sporting a 266MHz Cyrix CPU, 64MB RAM and a CD-ROM drive. The front of the unit sports a power and reset button. On the back, you'll find two USB ports, two PS/2 ports for the NIC keyboard and mouse, a video out connector for a monitor, an RJ-45 jack for built-in 10MBps Ethernet, two RJ-11 jacks for the NIC's built-in 56K modem and a (curiously) nonfunctional joystick port. You can orient the NIC horizontally or, if you prefer, vertically using two detachable feet.

Getting to Work

My initial intent was to hook the new NIC to my LAN, as I use Linux, DSL and IP masquerading for Internet access. I unpacked the NIC, plugged in all the cables, connected the NIC to a nearby hub and turned on the computer. I then inserted the NIC CD-ROM, pressed Enter, saw an initial splash screen on my spare monitor and then...nothing! The screen cleared, and a small, flat blinking cursor appeared on the upper-left corner of my monitor as the CD-ROM spun down.

With a sinking feeling in my gut, I called NIC's technical support line and was connected to a help technician in less than ten minutes. After describing my problem, Rod, the technician, told me to turn off the NIC, then power it up and tap the keyboard's Delete key twice to access the NIC's built-in Award BIOS. After loading the default CMOS and system setups and saving the changes, the NIC rebooted, brought up Linux and displayed its initial configuration screen in 65 seconds. You may need to do this if you change the NIC's monitors between boot-ups.

When you first start the NIC, the system will boot Linux, load X11 and bring up a desktop running Netscape with a configuration screen, as shown in Figure 1.

Figure 1. The NIC's default application and configuration tool is Netscape Navigator, version 4.72.

There are a number of ways to access the Internet using the NIC. You can set up a free Internet account with NetZero using the built-in modem, configure the NIC to dial up and connect with your current ISP, or configure the NIC to use your LAN and a default gateway for Internet access or use as an X11 terminal. For those unfortunate users without local calling access to an ISP or NetZero, you can also set up the NIC to access BamNet at $0.06 US per minute. When you start the setup, you'll see a dialog, as shown in Figure 2.

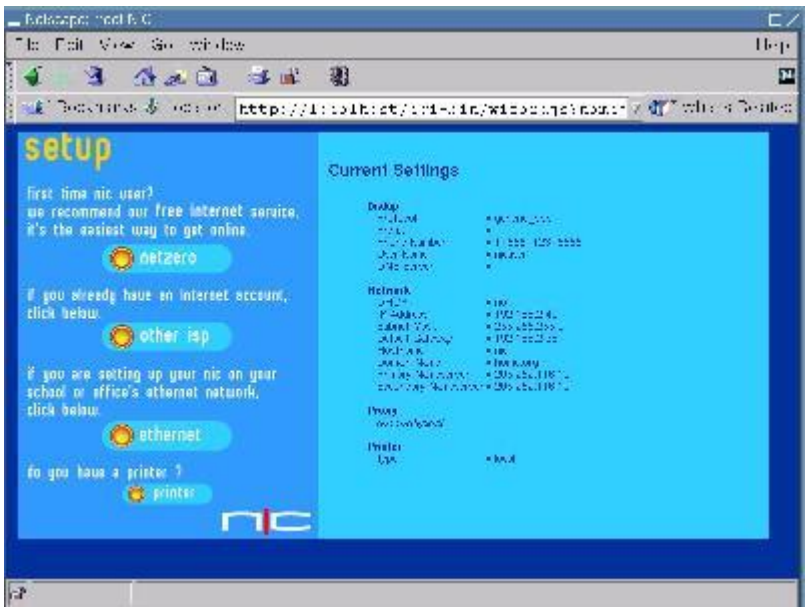


Figure 2. You can use NetZero's free service, your current ISP or an existing Ethernet LAN for Internet access.

After making sure a phone line was plugged into the back of the NIC, I then signed up for NetZero's free Internet service. After filling out several dialogs, the NIC dialed into NetZero and I was surfing. This took about about 15

minutes. Unfortunately, NetZero requires a floating advertising bar on your screen, or you'll be disconnected.

I also connected using my local backup ISP and was able to create the account and dial in within a few minutes. Connection speed was nearly 56KBps using the NIC's internal modem.

Setting up Ethernet and using a default gateway on my LAN gave me Internet access in less than 30 seconds. All settings are saved in the NIC's flash memory. You should know that the NIC's system does not offer Netscape Messenger for mail or Netscape Discussions for browsing Usenet news. You'll also find that the Preferences menu item under the Edit menu is grayed out and inaccessible. To use electronic mail or other services, you'll need to use remote web sites or local server software.

Sound is supported, and Real's RealPlayer G2 for Linux is included. This means you can browse to your favorite Internet news radio, TV or movie sites and hear stereo sound. The NIC's external speakers are small but amplified and feature a 3-D sound push button.

The fact that the NIC only offers Netscape and Internet Relay Chat when used as a basic Internet workstation could be viewed as a limitation. However, if you select the NIC's tools button on the local “Meet the NIC” web page (http://localhost), you'll find a button labeled “Tools” that brings up a desktop window with folders for a Windows Citrix client, assorted solitaire and board games, a secure shell terminal, Telnet terminal, ATT's vncviewer client and a simple IRC client. This is the extent of the software available on the NIC's CD-ROM.

I was also able to quickly turn my NIC into a networked X11 workstion and run clients from my server by first clicking the NIC's Xhost+ utility, starting a telnet session, then exporting the NIC's DISPLAY variable like this (I named the NIC “nic”, of course):

```
$ export DISPLAY=nic:0
```

I then ran numerous clients, as shown in Figure 3.

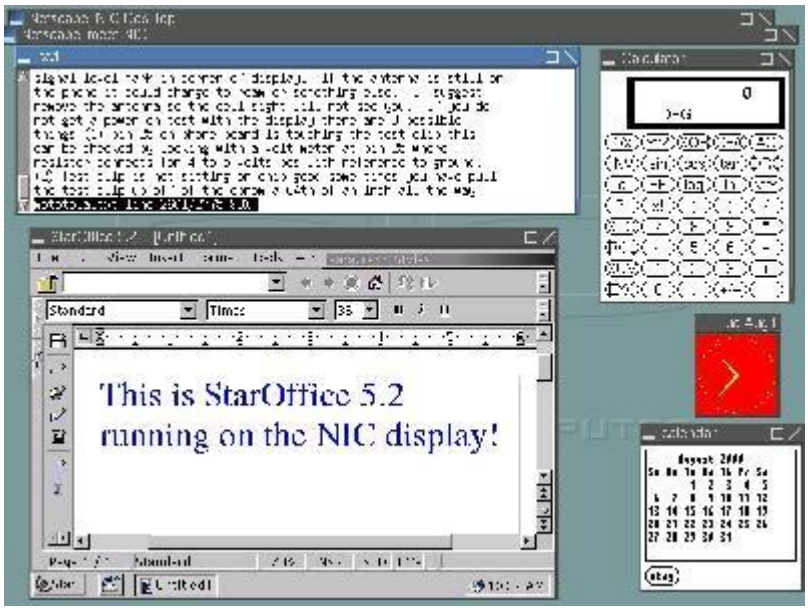


Figure 3. The NIC also performs admirably as a networked workstation.

Under the Hood

If you have the temerity to undo three screws on the back of the NIC, you can quickly access its innards, as shown in Figure 4. A quick peek shows a single-board computer, fan-cooled CPU, small power supply (with fan) and a CD-ROM drive. A 64MB PC100 DIMM provides system memory. A flash memory chip is used to save configuration settings and other items. Ethernet is provided by an SiS 900 chip set.


Figure 4. The NIC is a simple, single-board computer with few parts.

The NIC is a bargain considering the current street price of a single 64MB PC100 DIMM. Compare the NIC to Compaq's $499 iPAQ, IBM's $699 NetVista and ClearCube's $1,395 C3, and you'll see that the NIC is the least expensive, uses the same or smaller footprint and only lacks a hard drive. Linux hardware hackers will want to take a much closer look at the NIC's Award BIOS and IDE interface. Software wizards may want to explore modifying the Linux CD-ROM

and perhaps building a custom system to support external storage USB devices. One could, at the very least, burn a new CD with additional X11 clients.

Hacking the NIC

Obviously, the NIC is most easily used in a straightforward manner, either as a dial-up Internet appliance or a browser station attached to an existing LAN. Aside from playing some solitaire or pegboard games, the NIC's main function is to provide Netscape as a window to the Web.

The first thing you'll want to do is mount the NIC's CD-ROM in another computer and take a peek at the 200MB Linux file system. Don't bother booting the CD on another computer—you'll only get a kernel panic for your effort. After perusing the CD, you'll soon see that the NIC runs a Linux system stripped of nearly all software and services. Does this mean that you can't get a bit more functionality out of the system or modify the default folders and software?

Certainly not! After looking around, I found that I was able to launch nearly any X11 client from the CD-ROM while the NIC was running by using Netscape and a convenient launching client named **launchapp** in Netscape's URL field. For example, to pop up an rxvt terminal window with root access, you can use a URL like this:

```
http://localhost/cgi-bin/launchapp?/usr/X11R6/bin/rxvt
```

By the way, although not initially obvious, windows under the default Blackbox window manager are resizable. You have to put your pointer on the extreme lower-right corner of a client's window, then click and drag to resize. The grab area is small—about 20 pixels high by five pixels wide.

Once I had a root access terminal window, it was easy to start poking around while the NIC's file system was running. I next turned my attention to the NIC's desktop features. The NIC's tools folder is stored in flash memory under the /flash directory. Under the /flash/desktop directory, I found directories containing the Blackbox configuration files for the default games, clients and other utilities.

All I then needed to do was create my own entries to add additional programs to the desktop. However, I searched the file system in vain for any text editor, such as pico, jed or vi. Would any Linux hacker give up at this point?

Definitely not! Every Linux system includes a text editor, even if no text editors are installed. I navigated to the /flash/desktop/XTerminal directory, then used the cat command, along with output redirection, to create a desktop entry for the rxvt client:

```
# cat >rxvt.desktop<\n>
desktop_entry:
        name = rxvt
        icon = /img/telnet.gif
        comment = rxvt
        exec = /usr/X11R6/bin/rxvt
        terminal = false
        type = application
```

After pressing Enter at the last line, I then pressed Ctrl+D to save the file. Reopening the tools and desktop folder revealed the new entry. But what happens if you make a mistake or misconfigure your NIC's flash memory?

Don't worry. Just be happy that there's a “special” cgi-bin script you can use to upgrade or reset your NIC to factory status. Use the following undocumented URL:

```
http://localhost/special.html
```

You'll see a screen that allows you to update the system using a CD-ROM from The New Internet Computer Company or wipe your system clean of its configuration.

What's Missing?

Although the current NIC software distribution is labelled version 1.1, the only supported USB device happens to be the only supported printer—the Epson Stylus Color 740 printer. Considering the unused disk space available on the CD-ROM, I'd expect expanded printer support and many additional games, utilities or X11 clients. Also missing is a working help system or even a local copy of the users guide in HTML. Considering that the main NIC application is Netscape, the default home page should at least be set to an index for a small help system. The default Blackbox root menu should be modified to provide additional virtual desktops and window handling. Keeping in the spirit of open source, The New Internet Computer Company graciously provides links to all the source and patches used to build the NIC's CD-ROM. Browse to www.thinknic.com/gpl.html. You'll find links to every software package, including a link to an ISO9660 image of an updated version 1.2 system CD-ROM!

Despite some small initial limitations, the fact is that the NIC works very well. This appliance represents the first, best and least expensive of the new breed of affordable Internet and network appliance computers. Viewed in the context of its design, this device is a bargain. And considering that the NIC runs Linux, this device offers a tantalizing opportunity for Linux hardware and software hackers.

The Good/The Bad

Bill Ball is the author of numerous books about Linux but still doesn't know what to do with his multiple copies of shrink-wrapped Microsoft operating system software and CD-ROMs. He is a member of the Northern Virginia Linux Users Group.

Archive Index Issue Table of Contents

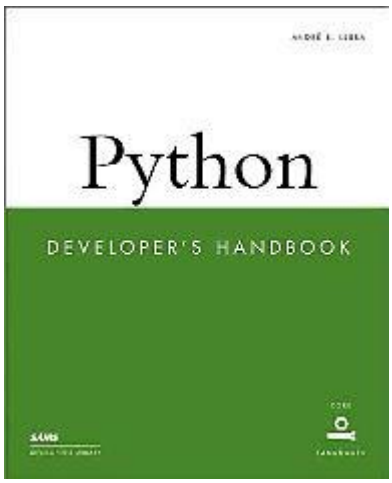
Advanced search

Python Developer's Handbook

Phil Hughes

Issue #82, February 2001

Yes, we do need this 800 page monster.



- Author: André S. Lessa
- Publisher: SAMS
- URL: http://www.mcp.com/sams/
- Price: $44.99 US
- ISBN: 0-67231-9942

Buy this book today!

- Reviewer: Phil Hughes

Is there a need for another book on Python? That was the question I asked myself when I picked up this 800-page monster. My conclusion is, yes, there is.

This is one of those rare cases when the book title seems to match the book content. I certainly consider the book a handbook, and it is udoubtedly aimed at developers.

The book is divided into seven parts, each one of which contains from one to five chapters. The parts are titled “Basic Programming”, “Advanced Programming”, “Network Programming”, “Graphical Interfaces”, “Developing with Python”, “Python and Java” and “Appendices”.

Part one is 200 pages long and gets you up and going. After a quick introduction that includes comparative information showing how Python stacks up against C/C++, Perl, Tcl, Smalltalk and Java, the chapters go on to cover the language itself, libraries and exception handling. One particularly good part of this coverage is that functions specific to particular platforms (UNIX, IRIX, Sun OS, MS Windows and Macintosh) are broken out and covered. Even if you only intend to develop for one platform, understanding the specifics of all platforms is useful.

I particularly like the presentation of the material in this part. For example, descriptions of functions include a basic explanation and the syntax but are also generally followed by an example. If the concept is complicated, more space is dedicated to the example.

Part two begins where part one leaves off. It covers “Extending and Embedding Python”, “Objects Interfacing and Distribution”, “Working with Databases” and “Other Advanced Topics”. “Extending” does an excellent job of showing you how to integrate C code with your Python project. Again, liberal use of examples makes everything very clear. “Objects” deals primarily with COM, using examples to keep you on track.

“Databases” covers the use of flat files, DBM, Python's own marshal and shelve, and SQL. This section is fairly short and doesn't go into real detail, but choices such as PostgreSQL, Gadfly and MySQL are introduced, and you are given URLs to find more information. I don't consider this last part a shortcoming of the book as database design and access could (and should) be a book of its own.

“Other Advanced Topics” addresses image manipulation, sound, regular expressions and threads. This chapter has enough information for a serious programmer to get going but could be a little brief for someone new to these concepts. The chapter ends with examples that illustrate the concepts that were presented.

Part three, with over 150 pages, has a lot of meat (tofu?). After a basic introduction to networking that covers HTTP, FTP, SMTP and more, you are off to web development. This chapter covers web server configuration and then goes on to talk about applications such as Zope and Mailman and site management tools. The next chapter is dedicated to scripting and addresses such issues as security, sessions and cookies. The final chapter of part three

deals with data manipulation and covers an assortment of topics from XML to PythonPoint.

Part four consists of two chapters. The first chapter introduces graphical interfaces including stdwin, PyKDE, PyGTK and a lot more. The second chapter deals specifically with Tkinter and really gets down to the nuts and bolts.

Part five takes you back from the low-level design stuff and looks at the concepts of development. Again, there is thorough coverage of development strategy, integrated development environments and development and debugging tools on a variety of platforms.

The last body part is on JPython. JPython is a new implementation of the Python interpreter and libraries written in Java. Coverage seems complete.

The final part includes the appendices of the book. The first is documentation on the Python/C API. Coverage is thorough. Next is a chapter on platform specifics. It covers Win32, MacOS, UNIX, OS/2 Windows 3.1, DOS, BeOS, VMS, Psion, Windows CE and even a clue for anything that might have been missed. The final chapter is just the related licenses and copyright information.

Besides covering the language, this book fills in all the blanks. When something is difficult, examples are used to help you out. When the subject is larger than could be covered in this general book, references are given. All in all I found this to be an excellent and accurate book well worth the space it will take on a Python programmer's bookshelf.

Phil Hughes is the publisher of *Linux Journal* and *Embedded Linux Journal*.

Archive Index Issue Table of Contents

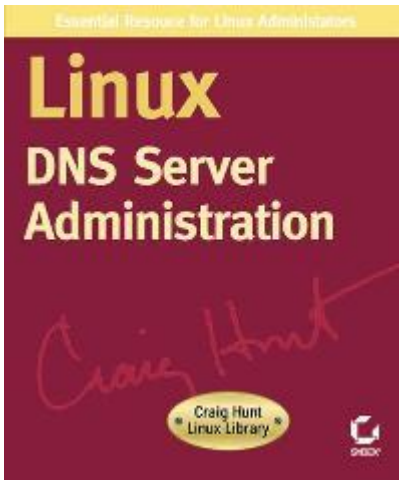
Advanced search

Linux DNS Server Administration

Ralph Krause

Issue #82, February 2001

Linux DNS Server Administration by Craig Hunt explains how DNS works and details the steps necessary to configure it.



- Author: Craig Hunt
- Publisher: SYBEX, Inc.
- URL: http://www.sybex.com/
- Price: $39.00 US
- ISBN: 0-7821-2736-3
- Reviewer: Ralph Krause

The Domain Name Service (DNS) is an integral part of networking and the Internet. Linux distributions typically include BIND, the Berkeley Internet Name Domain software, which handles DNS. However, books about Linux usually don't provide much information on using or configuring DNS.

While configuring BIND can be complicated, it is not impossible. *Linux DNS Server Administration* by Craig Hunt explains how DNS works and details the steps necessary to configure it. Mr. Hunt is a noted TCP/IP and Linux expert,

and he has done a very good job of illuminating the inner workings of DNS and explaining how BIND works with Linux.

The book contains twelve chapters and is divided into four parts that explain how BIND and DNS work, basic DNS configuration, advanced DNS configuration and how to maintain DNS once it is running. Appendices and an index make up the rest of the book.

The book is well laid out, and information is easy to find. At the beginning of each section is an overview of the material in the following chapters. Each chapter begins with a brief introduction of the material that it will cover and ends with a summary. A listing of all the code and example files is on the inside of the front and back covers.

The first part of the book covers the DNS architecture, protocols and the BIND software. The /etc/hosts file is explained along with its uses and limitations. The DNS Hierarchy is introduced along with an explanation of domains, searching for domains and how queries are resolved. Datagrams are provided for DNS messages and an explanation on how DNS databases are synchronized is given. Next is a look at BIND installation and control. The section ends with instructions for determining your own DNS architecture requirements.

The second part covers DNS configuration and contains three chapters. The first chapter explains what the resolv.conf, host.conf and nsswitch.conf files are for and how they work. The next chapter details the configuration of caching and slave servers and provides suggestions on when to create them. The final chapter explains how to create a master server, the one that is the authority for a given domain and the most complicated. Each chapter contains sample configuration files along with an explanation of their contents.

Part three deals with advanced BIND configurations. Topics covered include how to create nondelegated subdomains within a zone, when to create delegated child zones and how to advertise network services. This section also includes information on tuning a DNS configuration for better performance. The section also introduces the dynamic DNS (DDNS) protocol that promises to eliminate the drudge work of DNS configuration by having the computer create records from the information available on the network.

The final part of the book is concerned with keeping a running DNS system healthy and secure. The first chapter in this section covers security and includes information on securing Linux, securing the DNS configuration, coexisting with a firewall and the DNS Security (DNSSEC) protocols. The next chapter explains how to test and troubleshoot DNS, using commands such as **host**, **dig** and

nslookup. The final chapter covers the BIND log files and explains how to configure logging to meet specialized needs.

The book also contains four appendices. Appendix A introduces the new features coming up in BIND 9 and contains brief installation instructions for using the Beta 2 release. Appendix B is a command reference for the named.conf file. Appendix C contains descriptions of all the 41 types of resource records that BIND supports. The final appendix explains how to configure a Network Information Service (NIS) server.

I found this book relatively easy to follow and understand and was able to set up a DNS server for my small network using it. The book provides illustrations, definitions and sample configuration files that clarify the text. Mr. Hunt does a very good job of explaining how DNS works and how to configure and maintain a DNS server with a good balance between general and technical information.

Ralph Krause lives in southeastern Michigan and has been using Linux for over two years. In addition to writing about Linux and FreeBSD, he also does computer consulting and creates web sites. He can be reached at rkrause@netperson.net.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #82, February 2001

Readers sound off.

In Response to Debian 2.2 Potato: Memorial to a Hacker

Who writes your “THE GOOD/THE BAD” box? Couldn't (s)he at least read the article this is supposed to summarize? For example, the article tells us how much better the install has gotten, and the box says “The worst install ever”. Hello? Anybody at home there?

Hmm. What does “kernel 2.4.0test8” do in a “Hardware Profile”? And why is it mentioned at all? Nothing in the rest of the article talks about that, and there's certainly no such pre-alpha kernel in the distribution.

I see that “several packages are in need of updating”, but no specific package is mentioned. Somehow, I'm not particularly surprised.

Now *this* is truely offensive. If you had bothered to actually look, you'd have noticed that none of the compatibility libraries are needed for Debian-compiled programs: they're there for compatibility with foreign binaries. There's even support for compiling to old libraries for people who need that, though libc5 support may finally vanish in the next version unless our users tell us that they *really* want us to retain it.

“There is a plan to remove the boot floppy altogether from future distributions”. Nope, no such plan. There *is* a plan to replace the installer on those floppies with a modularized version.

“Debians package management system is also incredibly cryptic.” Hmm, well, what I've seen of rpm so far certainly didn't impress me as any less cryptic, and though most Debian developers agree dselect needs replacement with something more modern, I've seen users praise it more than once. Of course,

that's all personal opinion, but my impression is that a number of people only find the stuff cryptic because they were told to expect it to be cryptic. Personally, I think a typical Windows-like GUI installer is horribly cryptic.

"...a URI, which is similar to an URL". Ouch. Short tutorial follows: A URI (Universal Resource Identifier) can be either a URN (Universal Resource Name) or a URL (Universal Resource Locator). It's a URN if it's "stable forever", such as urn:isbn:0-345-38851-8; it's a URL if it tells you how to find the named resource, such as http://www.linuxjournal.com/. It can be both.

You know there's a saying that one should judge publications by how they talk about a subject one is familiar with? By that measure, this issue makes for a pretty steep drop in credibility. OTOH, I've seen better articles before, so I'll just chalk this one up to a sort of one-off.

—MfG Kaikaih@khms.westfalen.de

Retro Linux

Don Marti's "Building the Ultimate Linux Workstation" brings back fond memories of a similar article about 17 years back. "80 Micro" ran a feature comparing a couple of custom-built screamers. At about $3,000 apiece, I could only dream about buying one of those state-of-the-art machines.

Those blazing hot TRS-80 Model-III-compatible systems boasted an unbelievably fast 5MHz Z80B CPU, 128K of bank-switched RAM (the Z80 had the on-chip hardware to address only 64K), and get this, a 5 Meg hard drive, enough to store most people's entire collection of floppies. Eat your heart out, fellas.

Those folks lucky enough to have one of these rockets stored in their attic need to start putting the pressure on for porting the kernel to the Z80 chip. For that matter, there is a shameful lack of drivers for such critically important legacy hardware as the Friden Flexowriter, magnetic core memory, reel-to-reel tape drives, the KSR-33 teletype terminal and the paper tape reader/punch (much better to save cartons of punched tape than all those tacky CDRs). Hey, get on the stick, Linus. We want retro-Linux and we want it now!

—Mendel Cooperthegrendel@theriver.com

Exploiting Explorer

Upon opening the November, 2000 issue of *Linux Journal* to page 199, I almost fell out of my chair. *Linux Journal*, the premier magazine for the Linux community, had an advertisement (for Axis Communications) with a screen

shot of a Microsoft Internet Explorer window. Is it really so much about the mighty dollar that you have no restrictions as to the content of your advertisers' ads? At a minimum, you could have made it look like a Netscape window. I truly enjoy *Linux Journal*, and usually read it in its entirety, on the same day it is delivered. The last thing I want to see is an advertisement for a product (that runs on just about any platform) being depicted running in the Windows environment.

Thanks for a great magazine,

—Jerry Readjerry@jread.net

Be Specific

Through my letter box came a smoking issue. I loved the cover and the feature articles on building an Ultimate Linux Workstation (or even a lower budget version). I coveted all of these configurations, which were better than the surplus Celeron-based system I'd just commissioned as my personal Linux workstation to replace a failed 486-based Linux box (don't laugh—that 486-based workstation gave better performance than many high-price Windows boxes).

As I read these articles though, a long-running frustration of mine resurfaced. Many years ago I studied AI and expert systems when the computing science literature was full of talk about a hardware configuration system at Digital (now Compaq). Their system, known as either R1 or xcon, would check that a hardware configuration was viable. If not the system would remind the engineering team what needed to be added and finally generate a detailed list for the installation engineer to follow so that boards were plugged into the correct bus slots. With all the hardware options for Linux I've long thought that a similar expert system was necessary.

Obviously such a system needs a database containing what motherboard has what features, what hardware is necessary for a server and what is needed for a low-end/high-end workstation, etc. Specific issues with hardware options could be highlighted, e.g., what EIDE drives to avoid and why, what ASICs don't work correctly, whether to use Intel or AMD processors, whether Alpha or PowerPC chips would be more appropriate, what video cards have only partial support in XFree86, what software need not be installed on a workstation and what is required on a server, how many fans are needed and where should they be mounted, even down to issuing reminders to the installer of applicable CERT advisories. Extensions could help configure Beowulf clusters, firewalls, diskless workstations, laptops and many other useful setups.

I wanted a summary at the end of the Ultimate Linux box article of the various suggested hardware configurations. A simple spreadsheet (gnumeric of course) helped to collate all that advice and price it. In that respect the following article on the cheaper workstation was a better source of information. But being able to feed these configurations into an R1/xcon-like system would have been really helpful—especially if it held up-to-date price information. Then I'd know which supplier to buy which components from so I could get that ultimate workstation at the lowest cost.

If anyone is working on such a configuration tool I'd love to hear from them and/or to read about it in *Linux Journal*.

—Trevor Jenkinstrevor@suneidesis.com

What about Tapeware?

I just finished reading the Readers' Choice Awards and find it interesting that no one mentioned Tapeware from Yosemite software for backup.

We have used both tar and BRU but find Tapeware to be far superior to either. It is a distributed application (in the class of Arkei, I suppose?) and runs seamlessly distributed across both Windows and Linux platforms. The GUI is QT-based, which makes it look and feel exactly the same between Windows and Linux.

Just thought I'd mention this product since no one seems to know it exists.

—Bud Millwoodbudm@weird-solutions.com

Uses for CueCat

Good job with the CueCat article!

Living in Europe, I wondered why so much buzz about a bar code scanner. Then I read on the Net how it is distributed and what the attached software does (or is capable of doing). It stinks!

I'm glad *LJ* supports this kind of backward engineering efforts by publishing both the method and the results. At least it makes life a bit more uncomfortable for privacy invaders!

On the other hand, such an inexpensive device with open-source drivers can be sold by the thousands for perfectly legitimate uses. I will buy one on my next trip to the US, I'm thinking about:

- Finally classifing our library and keeping track of the borrowed material.
- Doing the same with our music/data CDs.
- Classifing our medicines. With two kids we don't know what we have, what it's good for, when it expires and where we hid it last time!

—Carlos Vidalcarlos@tarkus.se

In Response to Arthur ("Letters", November 2000)

I rarely feel the need to reply to mail printed in mags, but yours was so full of misconceptions that I felt an overwhelming drive to set the record straight and educate you simultaneously.

First, regarding your statement that "(America) consumes so much...yet gives little back...." Put on some sunglasses so that you may open your eyes and take a good look around you! Electricity to power the computer upon which you wrote that misguided e-mail originated in the USA. The computer itself, what is today accepted as the "PC", came from here. The Internet upon which your garbage was transmitted—not to mention the underlying technology such as telephone equipment—originated in the USA. Television, the car you drove to work, and the list goes on and on! Compare that to the list of things contributed to the world by your little penal colony (all I could manage is Foster's Lager, fuzzy brown fruit and Crocodile Dundee) then do the math!

—Todd Ficht, Americanficht@ieee.org

Be Warned

I was just reading the Video section of the article "Building the Ultimate Linux Workstation" and came across a couple of quotes from Darryl Strauss that I thought shouldn't have been there. He states: "The up-and-coming board to watch is the Radeon"; and "Performance is about the same as the GeForce2, and they want to do open-source drivers. It's just not out for Linux yet"

I would like to focus on the "they want to do open-source drivers. It's just not out for Linux yet" part. Let me spare anyone from the hell that I have gone through with ATI in the past and I ask that a little research be done on ATI specifically before the readers and subscribers of *Linux Journal* are recalling where they got the idea to buy the damn thing to begin with.

ATI has promised 3-D drivers for their cards under Linux for almost two years now. When an actual 3-D driver that is usable (and when I say usable thats exactly what I mean; drivers that make quake3 actually run a bit faster than the couple pixels a second you get, try for yourself) has yet to be seen. UTAH-GLX is not developed by ATI or anyone they've hired. They've hired the guys/gals at Precision Insight sometime ago (http://www.precisioninsight.com/), and they (ATI) promised to have ATI 3-D drivers available in Q1 of 2000. Needless to say Q1 of 2000 came, and they had no usable drivers; Q2 came, still no drivers. Q3 came, you get the idea.

They consistently boast on their web site that they have 3-D drivers for the Rage 3-D pro cards, which is correct but it's not their drivers; and it's not supported by ATI. You can find them here (UTAH-GLX) utah-glx.sourceforge.net. The Rage 128 3-D drivers are available here at dri.sourceforge.net, which happens to be Precision Insight's DRI project. Mind the Precision Insight drivers don't do much and aren't usable. So whatever you do, PLEASE do not buy the ATI Radeon before the drivers are completed and usable. DO NOT make the same mistake I made when I bought this ATI rage fury. Personally I'll never buy another ATI card again; not because of the hardware, which is of high quality, but because they don't support their hardware with drivers.

—Christopher Warnerchristopher.warner@mvbms.com

Ada's Shortcomings

I work with Ada quite a bit at work and enjoyed seeing the article on Ada in the latest issue of *Linux Journal*. I have to agree that many programmers should take a serious look at the language. It is highly structured, and I would guarantee that novice programmers will decrease their debugging time and their frustrations tremendously by using the language. Many C++ programmers could learn a great deal of discipline sorely lacking in that language by taking the time to learn Ada.

With that said, I do have to take issue with some of the statements made in the article. First, Ada does not provide a full suite of operator overloading/ overriding capabilities. It does not allow overriding of the assignment operator nor the array indexing operator. Also, you cannot specify a return by reference in Ada. The latter two difficulties can be a great hassle when developing container classes, since it makes the specification used by clients clumsy. The language also does not support type promotions, which can again lead to unnecessarily clumsy interfaces.

Ada does not have inferred templates (known as “generics” in Ada). Generics must be instantiated explicitly, and each instantiation, even if based on the same parameters, represents a new class incompatible with other

instantiations. This can be a roadblock to software reuse. The language also lacks a notion of “protected” in the sense that C++ and Java have; nor does the language have a construct for declaring an object's attributes constant over the lifetime of that object. (You should see how JGNAT works around these issues when providing an Ada-equivalent API to the Java core classes; it's not pretty.)

Perhaps one of the biggest drawbacks to the language is that the compilation of the code requires it to be “bound” before it is linked. The binding process creates additional code needed to elaborate software modules (in the correct order). This prevents dynamic loading of code at runtime; in other words it prevents “plugins”. I believe technologies are now coming out to allow Ada plugins, but it is not clear that they faithfully adhere to the Ada standard.

In short, the article could have done a better job at pointing out some of the disadvantages as well as some of the advantages in programming in Ada. Do you have any plans for an article on Eiffel, another highly disciplined language?

—Johnjohn.rusnak@mindspring.com San Jose, CA

Errata

The opening paragraph in the January 2001 issue (81) of Doc Searls' Linux for Suits article, “The Morlock Market”, is actually a quote from Neal Stephenson and should have been separated from the article text and noted as such.

Listing 2 of John Hall's “A Crash Course in SDL”, issue 81, reads, beginning at line 15:

```
value= ((red >> fmt->Rloss) << fmt->Rshift) +
           ((green >> fmt->Gloss) << fmt->Gshift) +
           ((blue >> fmt->Bloss) << fmt->Bshift);
return value;
}
```

But should have read:

```
value = ((red > fmt->Rloss) << fmt->Rshift) +
           ((green > fmt->Gloss) << fmt->Gshift) +
           ((blue > fmt->Bloss) << fmt->Bshift);
return value;
}
```

Not to worry, the listing is in correct form on our ftp site.

—Editor

Archive Index Issue Table of Contents

Advanced search

UpFRONT

Doc Searls

Issue #82, February 2001

Stop the Presses, *LJ* Index and more.

FAST Times

A year ago, not many people knew about Google. Being the search engine both by and for the Linux community, now everybody knows about Google. It doesn't hurt that Google now powers Yahoo's Web searches as well as its own.

While Google remains a fine search engine, the company's decision to patent its search methods hasn't sat well with those in the Linux community who don't cotton to software patents, which include many in the Free and Open Source Software communities.

Well, there are also other search engines with Linux and UNIX credentials. One is Fast Search and Transfer ASA (FAST) http://www.alltheweb.com/, a Norwegian company with offices in the US and a partnership with Dell. The two companies jointly and publicly intend to build the world's largest and deepest search engine.

Early last year, Lycos made a substantial investment in FAST and now co-brands FAST's four basic search engines: FAST Web Search, FAST FTP Search, FAST MP3 Search and FAST MultiMedia Search (all of which can be found at www.alltheweb.com and www.lycos.com/—they use the same engines).

FAST's engines run on FreeBSD and are reportedly developed on a mix of FreeBSD and Linux machines. In fact, FAST's first engine, FTPsearch, was developed under the Free Software Foundation's GPL. You can still download the GPL version of that software at ftp://ftpsearch.ntnu.no/pub/ftpsearch/. Search results are presented by Apache and PHP.

We also understand that some of FAST's people have been involved in PHP's development for a long time, and many of FAST's R&D people in Norway come

from one UNIX-oriented computer club at the University in Trodheim. It's called “Programvareverkstedet”, or PVV http://www.pvv.org/.

The products FAST sells are closed-source along with the search engine itself, which is also the case for every other search engine at this point (or at least that we know of—correct us if we're wrong).

For more about FAST's technologies, click the “Technology” tab on the company's home page.

In another significant search engine development, Yahoo began in November to charge businesses to hurry their listings into Yahoo's “Business to Business” and “Shopping and Services” areas within the “Business and Economy” category. For $199, Yahoo's Business Express program fast-tracks submissions for review and possible inclusion in Yahoo's listings in either of those two areas. According to the FAQ docs.yahoo.com/info/suggest/faq.html, “...any site submitted to these areas will be reviewed and either added or denied within seven business days. If your site is denied, you will be told why and will have a chance to appeal the decision.”

Meanwhile, the Open Directory Project (http://www.dmoz.org/) continues to grow at an explosive rate. A cursory set of searches shows the two services are highly competitive. The question now is, how do they scale?

There's not much you can do to help Yahoo other than work for the company or pay for a listing. But there's a lot you can do to help the Open Directory Project—mainly as an editor. Just navigate down to a topic that obsesses you and sign up to become an editor through the link on that page.

Copywright, Guthrie Style

> When Woody Guthrie was singing hillbilly songs on a little Los Angeles radio station in the late 1930s, he used to mail out a small mimeographed songbook to listeners who wanted the words to his songs. On the bottom of one page appeared the following: “This song is Copyrighted in U.S., under Seal of Copyright # 154085, for a period of 28 years, and anybody caught singin it without our permission, will be mighty good friends of ourn, cause we don't give a dern. Publish it. Write it. Sing it. Swing to it. Yodel it. We wrote it, that's all we wanted to do.” —Pete Seeger, June 1967

LJ History

In the February 1995 issue of *Linux Journal* Belinda Frazier reports on Comdex 1995 and its Linux presence:

> ...there usually isn't much about UNIX at Comdex. This year however, I was very pleased to find Linux represented at two booths at the show. Both Yggdrasil Computing, Inc. and Morse Telecommunication had Linux in their companies' banner.

In comparison, at the Fall Comdex, the number of exhibitors in the Linux Business Expo section was in the neighborhood of 500.

LJ INDEX—February 2001

1. Lines of code in the average electronic toothbrush: **3,000**
2. Millions of billions of calculations per second in the human brain: **20**
3. Billions of embedded processors sold in 1998: **4.8**
4. Percentage of embedded processors intended for PCs in 1998: **2.5**
5. Number of embedded chips in the typical family car: **20**
6. Number of microprocessors in the typical American household: **40**
7. Trillions of dollars in projected on-line business-to-business sales in 2002: **1.3**
8. Increase in Americans' exposure to advertising between 1971 and 1991: **6x**
9. Percentage of stress-related visits to physicians: **75 to 90**
10. Increase in meat consumption during the last population doubling: **4x**
11. Species of diseases: **250**
12. Species of weeds: **220**
13. Percentage of grain fed to livestock: **40**
14. Number of trees it takes to print the Sunday *New York Times*: **3,200**
15. Millions of prisoners in the US in 2000: **1.3**
16. Percentage of Americans who will spend time in prison, if current incarceration rates continue: **5**
17. Percent chance for an African-American male in the US of going to jail: **28**

Source:

- 1-17: *Understanding*, the new book by Richard Saul Wurman. (www.understandingusa.com)

Linux Bytes Other Businesses

by Heather Mead

Prior to what Omaha Steaks (http://www/omahasteaks.com/) believed would be the start of their busiest on-line shopping time of year, the 2000 holiday

season, they decided to revamp their web site to accommodate an estimated one million customers. eOne Group (http://www.eonegroup.com/), an Omaha-based eCommerce software and service provider, worked with Omaha Steak's internal IT department to build a web site that could be hosted on-site, integrated into back-end systems as well as provide advance delivery schedules and multiple ship-to-address options.

As the basis of the new web site, eOne used their Java-based, hardware-independent development application called jCommerce. Being an open-architecture application, jCommerce runs on almost any platform and can be customized for each individual customer and situation. As jCommerce provided the database-driven features Omaha Steaks wanted, including easily customized XML and XHTML tags and accessible userinterfaces, the software choice seemed easy enough; the real decision centered on what platform to use.

Chad Bukowski, chief architect at eOne, characterizes Omaha Steaks as a traditional, family-owned and operated business since 1917. He was unsure how receptive the company would be of the open-source philosophy and business model Linux offered, the platform eOne actively supports for their jCommerce software. But he also knew that their final decision would be based on price and performance and not necessarily the name attached to the platform. The benchmark stress and load tests were done on RS/6000 M80 IBM systems, their current AS/400 system and Dell Intel boxes running Red Hat Linux 6.2. During the test, the AS/400 system slowed when the number of separate, simultaneous customer orders headed toward 50; the RS/6000 topped out around 150-200 orders. The Dell boxes, according to Bukowski, processed 250-400 orders with little taxation.

Impressed by the benchmark results of Linux, its scalability and because Omaha Steaks “wanted something that wasn't tied to IBM”, Bukowski said that they were eager to launch their new web site running on Linux. Jeff Carter, CTO of Omaha Steaks, said that four main considerations made Linux the most viable option: price performance of the Dell servers versus the IBM offerings ($8,000 per machine for Dell vs. $250,000 per machine for RS/6000, plus licensing fees); overall acceptance and performance of Linux in the Internet space; stability of Linux versus NT on the Intel platform; and easy integration of Linux with legacy systems.

The new site went live in fall 2000, after a 60-day setup period. So far, both Omaha Steaks and eOne are pleased with the results. Reboots are nonexistent, and Bukowski says the system “is singing right along”. Customers are satisfied with the simplified and versatile ordering procedure, and Omaha Steak employees have had no problems adapting to the new system. Carter

underlines the importance of having a system that can be maintained with internal resources: no Java programming is necessary to maintain the parameter-driven site. Of Omaha Steak's relationship with eOne Group, jCommerce and Linux, Carter says, “No one else in the marketplace had all of these thing to offer.”

Apache Extends the Plateau

Netcraft's monthly survey (www.netcraft.com) of hosts providing HTTP service has told the story of Apache's long-running leadership ever since the survey began in mid-1995 when Apache weighed in with a share of just a few percents. Late that year, Apache, NCSA and “other” were all tied at around 30%. This was also when Microsoft's IIS came on the scene. Since then Apache and IIS have gained and held the leading two positions, with Apache taking the majority share by a wide margin. While IIS overtook “other” to assume second place in December 1997, with about 20% of all sites surveyed, Apache was already moving past a 50% share.

The story told since late 1998 has been a pretty stable one. Apache holds a strong 59.67% (down from 60.02%). Microsoft is up from 19.56% to 20.17%. Sun's iPlanet (which includes all the old Netscape servers) is down from 7.15% to 6.92%. But those are just shares of a total that is up for all three. Here are the gains:

- Apache: up 590,305—from 12,705,194 to 13,295,499
- Microsoft: up 352,960—from 4,140,977 to 4,493,937
- iPlanet: up 27,169—from 1,514,106 to 1,541,275

The rising tide lifts even the most competitive boats.

The most interesting news coming out of Netcraft's latest survey concerns uptime. “Hosting locations with at least five popular sites queried several times each week are reported in order of average uptime, and a league table of the fifty sites with the highest uptimes on the Internet is also maintained”, Netcraft reports. The resulting graphs are pretty interesting. We see that Slashdot was rebooting almost daily going into late 1999 when uptime began to move past 60 days. One server has been up since May of this year (we're writing in November). Slashdot right now has a very respectable maximum uptime of almost 190 days and a 90-day moving average of over 120 days. Netcraft reports that Slashdot is running “Apache/1.3.12 (UNIX) mod_perl/1.24 on Linux”. By contrast, Microsoft has a max of 75.22 days and a 90-day moving average of 18.76, running “Microsoft-IIS/5.0 on Windows 2000”. To be fair, however, we can point to Starbucks, which now is past 215 days of uptime since moving to Windows 2000. This is also “the longest uptime of any Windows 2000 site we've

seen on the Internet". The overall record-holder is www.charite.de, which has been up 836.49 days or more than 2.5 years. It's running "Netscape-Commerce/1.1 on IRIX".

By far the largest hosting site is Exodus Communications, which has 548 sites at just one "netblock" in Santa Clara, California. The four longest-running servers are all for Zope and all run Linux. They're averaging an uptime of 384 days, which also happens to be the entire measured period. Three run "Zope/(unreleased version) ZServer/1.1b1". The other runs "Apache/1.3.4 (UNIX)". Also on the Exodus list are smellygig.com (Apache on Solaris, 277 days), cluetrain.com (Apache on Solaris, 258 days), mysql.com (Apache on Linux, 183 days), whitehouse.com (yes, the XXX site, Apache on BSD, 162 days) and slashcode.com (Apache on Linux, 190 days).

Another interesting Netcraft observation concerns "how good the vendors are at running their own kit". Among the list of hosting locations with the longest uptimes, VA Linux is in sixth place. Sun is at ninth. Microsoft is at 25th.

Netcraft's latest survey information is available at http://www.netcraft.com/survey/.

Say Where?

Disturbing Search Requests (DSR, http://www.searchrequests.weblogs.com/) is a "collaborative weblog dedicated to misleading search requests" that mines the veins of irony found in both site referrer logs and search-engine results derived from link-loving bots and engine algorithms that give maximum value to the most highly linked sites. So, DSR continues,

> If you write a weblog on a regular basis, chances are you're going to post quotes from other sites, opinions from other people, etc. But since weblogs are highly linked to and from, they get indexed very well by search engines. So, even if you only once wrote about your hamster, and on the same day mentioned you were wearing a three-piece suit, Google just might list you as No. 1 for "hamster suit". Now just imagine that you check your referrer logs and you find a query from a search engine, looking for "hamster suit". This is where this site kicks in.

For example, Dave Bug of Geeklife (http://www.geeklife.com/) found that "Where are all the nice girls?" was not only a search request in his log files, but brought up The Value of Psychotic Experience (deoxy.org/w_value.htm) as a result on Google.

At this writing (November 20), there seems to be agreement among threads both within and linked to DSR, that Google and Yahoo (which uses the Google engine) sometimes exclude DSR from its findings. So we just typed “Disturbing Search Results” into Google, punched the “I'm feeling lucky” button and went straight to the site. Guess this isn't one of those times.

A New Linux Site for the Spanish-Speaking Linux Community.

Piensa.Com Systems has launched *La Gaceta de Linux* at http://www.gacetadelinux.com/. This site is the host of the Spanish edition of our sister on-line publication *Linux Gazette*. *Gaceta de Linux* is focused on providing firsthand information to the global Spanish-speaking Linux community. The site is made possible by the efforts of bilingual, English-Spanish volunteer translators who choose and select original articles in English and create a high-quality translation in just a few days. “We are fully committed to the Latin-American and Spanish-speaking use of Linux, especially in real-life business applications,” commented Felipe Barousse, CEO and General Director of BCM Piensa.Com Systems. “We believe Linux gives us a tremendous opportunity to spread technology with minimal cost and extremely high reliability and success where it is not feasible to deploy other proprietary options. *La Gaceta de Linux* is a great vehicle for this endeavor. Also, by the time you read this, the Portuguese edition, called *Gazeta do Linux* should be on-line as well.”

Translations are made by volunteers, and all articles in fact, are reviewed for quality and content, always respecting the original author copyrights. They have registered users from more than 22 countries and more than 100 translated articles since *Gaceta*'s opening last June. To join, log onto www.gacetadelinux.com/register and help spread Linux in the Spanish and Portuguese-speaking regions of the world.

Copyright, Guthrie Style

Stop the Presses: the Shift from Why to Why Not

At Comdex Fall in Las Vegas, the North Hall was where the wireless and telecommunications companies were concentrated. In the midst of the Internet Appliance pavilion was a booth for Be, Inc., the operating system company that has recently retooled itself to go after the Internet appliance business. While Be was making a valiant effort, the big news was quietly evident in other booths all around the floor, where Linux had quietly soaked into everything.

A company called Internet Appliance showed racks of servers featuring advanced failover technologies, dynamic content screening, browser-based

setup and administration and other goodies one would hope to find in “a complete Data Center for less than $30,000”. Does it run on Linux? Yes. Is that fact a big deal? No.

While Be hustled for every bit of visibility it could muster, Linux was quietly demonstrating its new default status as commodity OS infrastructure. Here are just a few of the Linux-powered devices I encountered at the show:

- Sony VAIO C1 PictureBook notebook computer (running on a Transmeta Crusoe processor)
- IBM NetVista N2200 thin client
- Ericsson Nanorouter 2 communication gateway and server
- Nokia Mediascreen Multimedia terminals
- e-Appliance SuperScaler network appliances
- Agenda VR3 sub-Palm-size PDA
- Snap Servers—portable snap-together servers from Quantum
- ZFLinux Devices—Linux embedded on an X86 chip (and chip set)
- Gateway's Connected Touch Pad internet appliance (also with a Transmeta Crusoe), which won “Best Consumer Product” from ZDNet and CNet judges

When I spoke with these companies in their booths, most didn't even give me reasons why they built their boxes with Linux. It was as if I asked why they used a TCP/IP stack or plastic in their cases.

Embedded Linux has become viral. Even when it isn't the subject at hand, the fact of Linux's rapid spread into embedded devices still comes up. While I was talking with a couple of guys at the Micronas booth (Micronas is a German company whose multimedia chips are in pretty much everything), they told me about a company called NetGem that makes Linux-based appliances that use televisions as monitors—very big in Europe, they report.

No doubt Be has a lot to offer. But it has to be sold. At this point, embedding Linux isn't even much of a brainer anymore. At some point in recent history, the question shifted from “why” to “why not?” Clearly there are fewer reasons every day.

They Said It

The seven worst words in cyberspace are “You just don't get it, do you?”

—Bob Metcalfe

There's no Moore's Law for software.

—Ellen Ullman

Like constipated feces, on-line advertising giant DoubleClick Inc. and handheld Internet-content leader OmniSky Corp. have impacted themselves within the growing intestinal bolus of companies experimenting with sending ads to wireless gizmos.

—Tom Matrullo

We've embarked on the beginning of the last days of the age of oil.

—Mike Bowlin, Chairman/CEO, ARCO

Build-to-flip financing is not a business model, it is a risk cap model.

—Patrick Sweet

Every extension is an amputation.

—Marshall McLuhan

Talent is always conscious of its own abundance and does not object to sharing.

—A. Solzhenitsyn

If the automobile industry had made as much progress (as the computer industry) in the past fifty years, a car today would cost a hundredth of a cent and go faster than the speed of light.

—Ray Kurzweil

Linux has to be important to General Motors. Not just to gm.com.

—Larry Augustin, VA Linux

Where there's chaos there's opportunity.

—Bryan Sparks, Lineo

Without chaos there would be no fun and games.

—Jonah Kinata

Linux is quickly becoming the operating system of choice for any new device, and the silicon vendors are sponsoring that.

—Inder Singh, LynuxWorks

There is an explosion of new devices, and more and more the assumption is that new devices will communicate.

—Michael Tiemann, Red Hat

Where devices are connected, Linux shines.

—Bryan Sparks, Lineo

The cost of hardware, software and network bandwidth are going to zero. When all three go to zero, the embedded space becomes infinite. So there is no reason anything you own should not contain some kind of intelligence. Linux as software is already there.

—Michael Tiemann, Red Hat

Tech Tips from Our Tech Editor

If you're running IP Masquerading and want to make services on your internal network available from the outside, just install **rinetd** on the IP Masquerding box available at http://www.boutell.com/rinetd/. (Also available as a package; check your distribution's web site.)

If you let other people run Perl scripts on your machine, make a symlink from /usr/local/bin/perl to the real location of Perl so they don't complain, “Hey, you don't have Perl.”

Another thing that shouldn't make any difference but sometimes fixes weird problems: switch PCI cards around in their slots.

If you have a complicated **crontab**, do a **crontab -l > my_crontab_`date +%Y%m%d'** to save it in case you royally mess up a **crontab -e.**

vi Navigation

Use (,), [,], { and } for navigation marks in vi. The **%** command will move the cursor to the matching mark. For example, with

```
( some text [ some more stuff
another line
{  ]
} )
```

If you position the vi cursor on any of the marks, in command mode the % will move the cursor to the matching mark.

You can use this method in comments in a programming or scripting language to allow quick fast-forward and fast-backward over large blocks of code, as well as use it to find boundaries of functions in languages like C which use one of these marks to delimit functions.

Mapping

Map an unused key to change files in vi. For example, if your .exrc or .vimrc has **map , :e#^M** when you've edited two files, for example, by giving

```
vi file1 file2<Enter>
:n<Enter>
```

so that you're in file2 and in command mode, typing **,** (no Enter required) will switch you back to file1. Use it again, and you switch to file2 and so on.

Job Opening Trends

by Reginald Charney

The falling fortunes of the dot-coms and the uncertainty of the election results, even before the election, have affected job figures. While the number of jobs offered has fallen since April, the decline has leveled off. Chart #1 shows the normalized job trends over the last 11 months. Ordinarily, things that don't change are not very interesting. But in this case, the fact that things have leveled off is good news.

(Note: Chart #1 is normalized for the number of jobs in January of this year. That is, the number of openings in January 2000 has been taken as 1.00.)

Figure #1

Platform Demand

Over the last 18 months, a number of platforms have competed for dominance. One of the interesting aspects of this is how fast demand for the main platforms has been growing or shrinking. Demand is defined as the rate of acceleration/deceleration in the trend line for the period shown. From Chart #2, we can see that demand has been slowing for the older platforms, while the newer ones, like Windows 2000 and Linux, have accelerated faster than all others.

Figure #2

Again, the exception to the rule is Solaris. It is right up there with the newbies. This indicates that Sun is increasing its dominance of its markets. These demand lines are positive at the moment because of the long period of expansion. Over the last 30 days, demand for all platforms has followed the general trend and decelerated. However, the short term has not yet outweighed the long-term demand.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #82, February 2001

Our experts answer your technical questions.

Network config Tools

In Windows NT there is a command-line utility (**ipconfig**) that lets you see the current ip configuration. Is there a utility in Linux for this? —Skip Bigelow, sbigelow@aarp.org

Even though there are graphical tools to give the information you've asked (including Red Hat's **netcfg** command), you can always use **/sbin/ifconfig**. It will give you detailed information regarding all active interfaces (ethernet, ppp, loopback etc.). —Mario, mneto@argo.com.br

Sharing a Cable Modem

I have been browsing many different Linux web sites to look for any FAQ or guide on this topic: How to share a cable modem connection at home between a Linux and a Windows machine, which is where the cable modem installed. I would appreciate it if you would give me some pointers. —Samuel Fung, samfz@hotmail.com

I would move the cable modem to the Linux machine and share it with your other computers from there. Why? Because Windows has no provision, off the shelf, to serve as a router, enable security features such as packet filtering, masquerading, forwarding, etc., while Linux does all that quite naturally and quite well. You do not specify the cable modem you have, but I would suggest looking at http://www.linuxdoc.org/ for documents on networking and connecting network devices to your Linux box. After that, look at the how-to articles on connecting to an ISP. —Felipe Barousse, fbarousse@piensa.com

Booting without Messages

Is it possible to turn off the kernel boot-up messages? —Nicholas, vunch@pacific.net.sg

The easiest way is to set console=ttyS3,38400n8, or something similar, on the LILO command line to redirect console output to a serial port. —Marc Merlin, marc_bts@valinux.com

Root Compromise

When I tried to log in to my Linux box this morning, I was surprised to find out that I was no longer able to do this. The login prompt appears as usual, but when I type the user name and press Enter, instead of the password prompt a new login prompt appears. No messages appear except a line that says: **/var/hackr0x/login: No such file or directory**. This line disappears so quickly that I had to repeat the procedure of typing the user name a couple of times in order to decipher it. —Victor, victor@angolatelecom.com

Your machine was indeed compromised. At this point you don't want to fix your machine, you just want to get your data off and re-install it. You don't know what's been modified nor how. In cases where you can't log in at all, you can always boot with **linux init=/bin/bash** at the LILO prompt, and then do: **mount -wno remount/mount -a /etc/rc.d/init.d/network start** (if you want to back up data over the Net). You can also boot from a rescue floppy or CD. Once you get your machine re-installed, do not just connect it to the Internet again without securing it properly. Make sure you have all the updates installed; do not run any unnecesary dæmons, and firewall the machine if possible. —Marc Merlin, marc_bts@valinux.com

Every major distribution has an “announce” list for security updates. After you reinstall, get on the list for the distribution you run. Also, remove unused software—it's the cheapest, fastest security precaution you can take. —Don Marti, info@linuxjournal.com

Slash Notation for Netmasks

Nowadays I'm working with Linux firewalls, and I'm configuring one in a client organization. I found the following lines in the script that applies the rules of the firewall (IPCHAINS):

```
INT0="eth0"
IP0="192.168.1.125/24"
NET0="192.168.1.0"
```

What is “/24” in the IP number?

Also can I put two networks in the same variable? For example:

```
NET0="192.168.1.0,192.168.10.0"
```

—Fabio Losnak, fabiolosnak@yahoo.com

The “/24” in the IP number means the network 192.168.1.0 with a netmask of /24 or 255.255.255.0. You probably cannot put two networks in the same variable but that would really depend on the script that is parsing this. —Marc Merlin, marc_bts@valinux.com

Can't Unlink Files

As root, I cannot get rid of the following files; they should belong to the deb package r-base, but in this case they seem to be some kind of links:

```
pimento:/home/ottoz# ls -l /usr/lib/R/library/ts/latex/
........
br-xr-srw-    1 25955    26473    116,  32 mar 20  1987 beavers.tex
br-xrwSr--    1 8301     31084    114,  32 ott 12  2021 sunspot.tex
br-srw-rw-    1 29281    8302     116, 108 set 27  2031 ts.union.tex
```

I get a message like **cannot unlink. operation not permitted** —Odoardo Zecca, odoardo.zecca@galactica.it

You had some file system corruption. **chattr -i *.tex** should remove the incorrectly set immutable flag and let you delete the files. —Marc Merlin, marc_bts@valinux.com

Restricting E-Mail Accounts with Sendmail

I have a mail server (RH 6.2, Sendmail Single Switch) acting as a smart relay on our DMZ. Internally, we have a mail server (RH 7.0, Sendmail Single Switch) that acts as both an SMTP and POP3 server.

We need to be able to differentiate between local-only and WAN e-mail accounts. Local-only accounts would be limited to local delivery/receipt and WAN accounts would be granted access to the world for inbound and outbound mail.

To further complicate matters, all users should have the format of first.last@domain.com for e-mail addresses. Is there a method of doing this within the capabilities of Sendmail or, if not, what package(s) will allow me to do this? —Michael Phillips, mike.phillips@ieionline.com

There are many approaches to solve your riddle. For instance, an easy one would be to restrict e-mail relaying with the /etc/mail/access file on an client IP address basis. Your actual request is not a complex one, you just need a bit of a

planning on your network layout, the addressing scheme and a bit of tuning on the Sendmail side. Go to the http://www.sendmail.org/ site and look for all relaying-related documents. That will help you solve your requirements. —Felipe Barousse, fbarousse@piensa.com

Telnet Sessions Time Out

Is there a Telnet time-out setting on Linux? My sessions time out after about five minutes. —Jan Dubroca, jan.dubroca@delta-air.com

You are probably Telneting outside of your network, through an IP masquerading server that times out TCP connections after five minutes. On Linux, the fix for this (on the firewall) is:

```
# Fix the masquerading timeouts
#                  tcp     tcpfin  udp
ipchains -M -S     86400   60      120
```

—Marc Merlin, marc_bts@valinux.com

I don't believe that there is a tim-eout setting for Telnet. I assume that what's timing out is your shell. The shell time-out can be set in **/etc/profile**. My guess is you've got an entry that looks something like this:

```
TMOUT=300
```

The value here is in seconds. You can change this to give yourself more time or simply remove the line to disable shell time-out completely. —Paul Christensen, pchristensen@penguincomputing.com

Loading vmlinuz, Then Nothing

I used Red Hat 6.0 to install Linux. I booted the machine from CD-ROM successfully, and I pressed Enter after boot: This message appeared:

```
Loading initrd.img.....................
Loading vmlinuz........
```

Then the computer stopped.

When I booted my computer from the floppy disk (Win98 bootdisk), I ran the **/dosutils/autoboot** from the Red Hat CD-ROM. Unexpected, it appeared that I had installed Linux successfully and even configured the X Windows System well. In the end, the computer told me:

```
Congratulations, you have installed linux successfully,......
The system reboot....
```

And when it rebooted, this message appeared:

```
Loading linux.........
```

Then the computer stopped again.

I have also tried Red Hat 5.0, Bluepoint1.0 and 2.0, TurboLinux, Slackware. The results were all the same. WHY? —ekun, xx@public1.ptt.js.cn

Apparently, you can boot Linux from **loadlin** (which you did when you started the Linux install from Windows), but, for some unknown reason, it fails when you boot with LILO. One option is to do an install from Windows, like you already did, and then boot from the RH rescue CD-ROM. Copy your kernel (**in/boot**) to the Windows partition (which you will need to mount too). Copy and configure loadlin (you should have them on your RH CD-ROM), and use loadlin to boot Linux. A sample loadlin config from my system looks like this:

```
moremagic:/drv/c$ cat linux.bat
c:\linux\loadlin\loadlin @c:\linux\loadlin\boot
moremagic:/drv/c$ cat linux/loadlin/boot
c:\linux\loadlin\vmlinuz
root=/dev/sda6
ro
```

—Marc Merlin, marc_bts@valinux.com

I've seen this happen as the result of booting a kernel that's optimized for the wrong processor, but if this is happening right away after a fresh install (in fact, after EVERY install of any of a number of distributions) you most likely have a serious hardware problem. You should try different RAM if you have any available. I can't say for sure that the RAM is the culprit, but that's where I'd start. —Robert Connoy, rconnoy@penguincomputing.com

Recompiling the Kernel

I have installed an IDE Atapi Zip drive and need to know how to have Linux to find it. I have tried to recompile but get the following error:

```
Makefile Makefile: 213 arch/i386/Makefile:
No such file or directory
Makefile: 481 Rule make: No such file or directory
make *** No rule to make target Rules.make. Stop
```

—Bob Parry, robpar@telus.net

Did you read README in **/usr/src/linux**? You compile a kernel like this:

```
make menuconfig; make clean; make dep;
make install; make modules; make modules_install
```

More details can also be found here: www.linux.com/howto/Kernel-HOWTO.html. If you already have the right module compiled, **modprobe ide-floppy** should do the trick. —Marc Merlin, marc_bts@valinux.com

I Have No Dæmon, and I Must Print

When I try to send anything to the printer, I get **Job is queued, but cannot start dæmon**. I can print by sending text directly to the printer with **cat**, but it won't start the dæmon. lpc status shows no dæmons started. I've set up printers many times before, but I've never run into this problem. I've tried removing and reinstalling the lpr package and all of the other tricks I've read. Even—Jim Jerzycke, kq6ea@amsat.org

Make sure that the printer spooler dæmon is running. With SuSE you'll want to make sure that the file **/etc/rc.config** includes the line

```
START_LPD="yes"
```

If it is not there, or if it is set to no, you should make the change, and then run —Robert Connoy, rconnoy@penguincomputing.com

Archive Index Issue Table of Contents

Advanced search

New Products

Heather Mead

Issue #82, February 2001

X Windows System Ported for Intel IA-64, Best Linux 2000, Epitera Desktop with Kandu Genie and more.

X Windows System Ported for Intel IA-64

Metro Link Inc. announced a port of its X Window System products for use on Intel's new IA-64 Itanium processor. The commercial grade X server, Motif and OpenGL products are geared toward high-end workstations, such as those designing graphics animation, architectural renderings and scientific visualization. X Window System includes Metro-X, an enhanced X server replacement; Metro Motif Complete!, three coexisting Motif versions; Metro Media, video-in-a-window and MPEG playback; and Metro Extreme 3-D, a hardware accelerated implementation of OpenGL.

Contact: Metro Link, Inc., 5807 North Andrews Way, Fort Lauderdale, Florida 33309, 800-821-8315 (toll free), sales@metrolink.com http://www.metrolink.com/.

Best Linux 2000

Best Linux 2000, a distribution from SOT Finnish Software Engineering Ltd., is available for retail sale in a four CD-ROM box set. Geared toward the desktop user, it includes over 2,000 applications such as StarOffice, GIMP, CD writing

software, games, multimedia, databases and more. A graphical installation is used for setup, and it can run multiple OSes with a graphical boot menu.

Contact: Best Linux/SOT, 6975 Washington Avenue South, Suite 210, Edina, Minnesota 55439, 952-947-0822, http://www.bestlinux.net/.

Epitera Desktop with Kandu Genie

A new desktop environment is delivered by Epitera, an Israel-based company, targeted at the PC and Internet appliance user. It features automatic downloads and updates as well as remote access customer support. An animated genie character named Kandu guides users through a three-level help system.

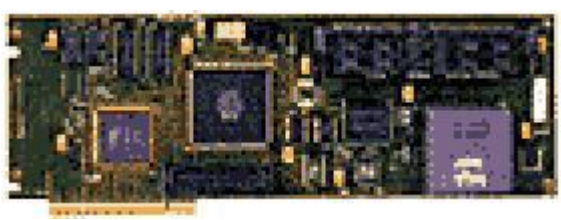
Contact: Epitera USA, 100 West 74th Street, Suite 4S, New York, New York 10023, 212-874-1198, jgundell@epitera.com, http://www.epitera.com/.

Oracle9i

Oracle9i offers a consolidated infrastructure of traditionally separate business technologies in a single, relational database and application server. Abilities include processing date from various locations (web sites, call centers, etc.) on one system; increased performance and scalability because processing and transformation is done in core databse; and fast implementation, low cost and easy manageability because fewer parts are required. The Application Server is currently available, and the Databse, Warehouse builder and BI Beans are scheduled for Spring 2001.

Contact: Oracle Corporation, 500 Oracle Parkway, Redwood Shores, California 94065, 650-506-7000, http://www.oracle.com/.

qX25 for Linux and PCI-bus ARTIC Cards



Quadron Corporation announces qX25 for Linux, an X.25 protocol package for PCI-bus ARTIC communication cards inside servers and workstations. qX25 is an X.25 development toolkit that enables users to create a custom solution for their particular communication requirements. Each ARTIC card has its own on-board processor and is a dedicated programmable communication computer residing within the host system. Three PCI-bus ARTIC cards are offered, ARTIC186 8-Port PCI Adapter, ARTIC186 Model II ISA/PCI Adapter and ARTIC186 X.25 ISA/PCI Adapter.

Contact: Quadron Corporation, 209 East Victoria Street, Santa Barbara, California 93101, 805-966-6424, info@quadron.com, http://www.quadron.com/.

E-Disk SCW35

A hot swappable solid-state storage solution, the E-Disk SCW35, has been released by BiTMICRO NETWORKS. The E-Disk is designed for solid-state disk storage with optimum performance, reliability and uptime. It can be inserted and removed from a server, RAID system or JBOD environment without powering down. The SCW35 is a drop-in replacement for any standard single-ended SCSI 3.5-inch hard drive, dlash disk or solid-state disk drive with an 80-pin SCA-2 connector.

Contact: BiTMICRO NETWORKS, Inc., 5550 Northport Loop East, Fremont, California 94538-6481, 510-74E-Disk, info@bitmicro.com, http://www.bitmicro.com/.

XPloy 2.0

XPloy 2.0, an e-business systems management tool from Trustix AS, is now available for secure and remote management of server resources from a single point. Updates from the previous version include the ability to load only the modules in use into the running environment, a load-balancing module and a systems operations monitor. XPloy 2.0 is a subscription service and rates are $39 per month, per server, based on a 3-year subscription.

Contact: Trustix AS, PO Box 1245 Leiv Eiriksson Senter, N-7462 Trondheim, Norway, +47-73-54-50-54, http://www.trustix.com/.

Communicado Fax

Communicado is a fax suite that combines application-based print-to-fax from multiplatform clients with a multiline server platform, allowing any platform to fax from any application. Features include: broad hardware and application compatibility; built-in address books, fax viewer with thumbnails and cover pages; and forwarding, reporting and accounting capabilities. There is also support for multiple inbound and outbound lines.

Contact: Merlin Software Technologies International, Inc., Suite 200 - 4199 Lougheed Highway, Burnaby, BC, Canada V5G 3Y6, 877-988-7227 (toll-free), info@merlinsoftech.com; http://www.merlinsoftech.com/.

Interchange 4.6

Interchange 4.6 is a software suite for managing e-commerce sites that is customizable for individual customer's requirements. Interchange is geared for medium-sized businesses and offers intuitive content management functionality. Interchanges software covers product presentation, merchandising, transaction management, reporting, customer management, backend integration and system administration. Interchange 4.6 is combination of the popular Tallyman and Minivend software packages. Demonstrations and product tours are available on the web site.

Contact: Akopia, Inc., 11480 Sunset Hills Road, Suite 200 East Reston, VA 20190, 703-456-2900, info@akopia.com, http://www.akopia.com/.

Archive Index Issue Table of Contents

Advanced search