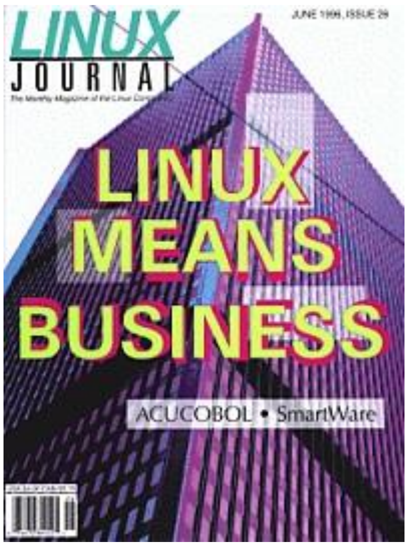
Linux Journal Issue #26/June 1996



Money Management in Linux

Curt Olson

Issue #26, June 1996

Personal finance programs have been popular in the DOS and windows environments for years, and so it is not surprising that until about a year ago, many new Linux users wished for a good personal finance program for Linux—but there simply wasn't one available. One creative user decided to do something about that.

Have you ever wondered where all your money goes? Do you get a nice big check on payday, but when it comes time to pay the bills, your wallet is looking rather thin? It seems there is never any money left over to save for the future. There comes a time in every person's life when it becomes obvious that they need to take control of their finances.

When I began dating the woman who is now my wife, I noticed that my money began disappearing just a little bit faster than before. As our relationship grew, so did the rate my money disappeared... This trend continued until it climaxed in the two weeks surrounding our wedding, when all the bills came due. I have never seen money disappear so fast.

Little did I know that this was only the tip of the iceberg. It seems that as life goes on my money gets stretched thinner and thinner. Now I find myself in the midst of purchasing a house, making car payments, buying dog food, and wondering how I am ever going to support my computer habit.

Needless to say, my wife and I needed a way to keep track of where our money was coming from and going to—and we needed a way to plan for our future expenses.

There are many excellent personal finance programs available at little or no cost. Unfortunately, none of the ones I found could be conveniently run under Linux. So in August of 1994 I said to myself, “How hard could it be? I will just whip up a quick little check book balancing program that runs under Unix.”

Well, as with most “little” projects, it turned out to be a bit more work than I expected. This little program turned into CBB. CBB is copyrighted under the GNU General Public License, so it is completely free. It is my humble contribution to the world of free software—a small payment for all the wonderful software others have so kindly made available for free.

What is CBB?

CBB is a personal check book balancing utility for Unix and X11. It was primarily developed under Linux, but runs equally well under most flavors of Unix. CBB is written entirely in Perl and Tcl/Tk, so it is portable and extensible. It is a program for anyone who would like to track their income and expenses, balance their checkbook, and manage their money. Any other use (such as lining for a cat's litter box) is not supported or recommended.

CBB is an open, extensible program written entirely in scripts. It utilizes a simple ASCII data file format. In addition, CBB provides a simple interface for users to add their own reports and graphs without modifying any of the CBB source code. In the future, I plan to create a simple interface to other external modules so that other people may provide plug-ins to increase CBB's functionality.

CBB (if you haven't guessed already) stands for the Check Book Balancer. This name illustrates the extreme amount of creativeness that is inherent in us computer nerds. My wife, who is not a computer nerd, suggested “In Cheque—Putting the balance in your budget.”

Feature List and Description

- Ability to create, edit and delete transactions. Automatically calculates the running balance.
- Many input accelerators to reduce data entry work. For instance, the **+** and **-** keys will increment and decrement the value in the date and check number fields. Transactions will be automatically completed by typing the first few characters of the description and pressing **<TAB>**. The rest of the transaction will be filled in from a matching transaction.
- Each transaction is assigned a category such as “entertainment” or “food”.
- Ability to split the amount of a transaction across multiple categories.
- Able to undo the last transaction insert, edit, or delete.
- Handles multiple accounts.
- Handles transfers between accounts.

- Performs account balancing: i.e., the ability to enter a statement's starting and ending balances, select uncleared transactions, verify that *start balance* + *transactions* = *end balance*, and clear all selected transactions.
- Contains a simple interface for external “user written” reports and graphs. Currently “ships” with three reports and one graph.
- Ability to import from and export to the Quicken export file format. This feature has not been extensively tested, but should provide the ability to move back and forth freely between Quicken and CBB.
- Able to handle recurring transactions. One of the contributed scripts adds this functionality.
- Support for the international date format, i.e., 30.01.68 (DD.MM.YY).
- An “auto save” function for the ASCII format data files. (This can save you when someone logs you out without asking.)
- The current X-Windows selection can be pasted into any entry field. Likewise a selected piece of text in CBB can be pasted into other X-Windows applications.
- Extensive reference manual is available in LaTeX, dvi, PostScript, or on-line locally in HTML format. The manual is also available at www.me.umn.edu/home/clolson/cbb/cbb-man/cbb-man.html.

Prerequisites and Installation

Before installing CBB, you should make sure you have Perl version 4.036 or later, and Tk version 4.0 or later installed on your system.

Installing CBB is very straightforward:

1. Unzip and untar the distribution file, and change to the newly created cbb-0.*??* directory.
2. Type **make install**.

You will be prompted for 4 items:

- The location of your Perl binary.
- The location of your wish4.0 binary.
- Where to install the CBB executable scripts.
- Where to install all the associated CBB files.

CBB attempts to present you with reasonable defaults for all of these items.

When you have finished answering these four questions, CBB will be installed.

Short Tutorial

So, you want to go for a little test drive? Want to see how, or if, this thing works? Want to send me a 21 inch monitor or the DOS drivers for my ancient Sony PAS16-SCSI CD-ROM drive? Well, read on...

The following procedure will lead you through the process of creating a new account, importing some data, editing transactions, and balancing your account.

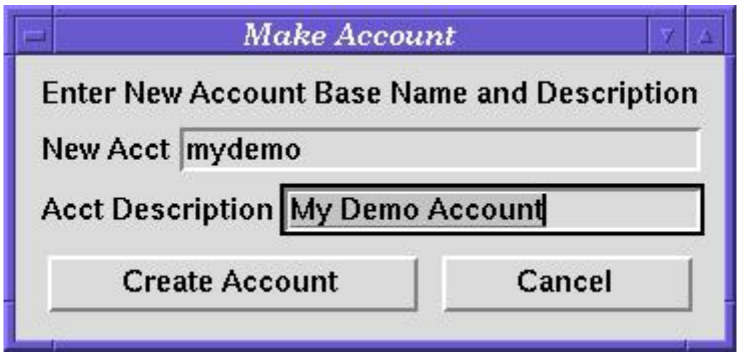
First, here is the “one-paragraph” version of the tutorial. To use CBB, first create the directory where you would like to keep your accounts. Then, change to this directory, run CBB, and create the accounts. You may import the default account categories at any time. Finally, load the desired account (if it is not already loaded) and create, delete, and edit transactions to your heart's content, while printing reports, viewing graphs, and so on. When your bank statement arrives in the mail, balance the account.

For a more detailed tutorial, read on...

Create a demo Account

After installing CBB, the first thing you need to do is create an account.

- Run CBB by typing **cbb**.
- Select **Make New Account ...** from the **File** menu.
- Enter an account name (e.g., **my-demo.cbb**) and an account description (e.g., **My Demo Account**).



Enter New Account Base Name and Description
New Acct: mydemo
Acct Description: My Demo Account

Figure 1. Making Account

- Click on the **Create Account** button. Your new account will be created and added to the master account list at the bottom of the main window. The name of the account will also be added to the category list (e.g.. **[my-demo]**). This name is used to specify transfers between accounts.

- You will be warned about not having a categories file. This is perfectly normal at this point. An empty categories file will be created for you.
- If you want to use the default categories, pull down the **Functions** menu and select **Categories -> Add Default Categories** to create the default categories.

Import Some Data

Once you have created an account, it is time to enter a few transactions. You can import some sample data to save your fingers from the brutalities of typing. The CBB distribution comes with some sample data for this tutorial. Otherwise, feel free to skip this section and enter your own transactions.

- Select **Import QIF File ...** from the **File** menu.
- You will be presented with a file selection dialog box. Navigate to the CBB distribution directory. Beneath the distribution directory is a **demo** directory. Go to the **demo** directory and find the file named **demo.qif**. Double click on this file to select it and import it.

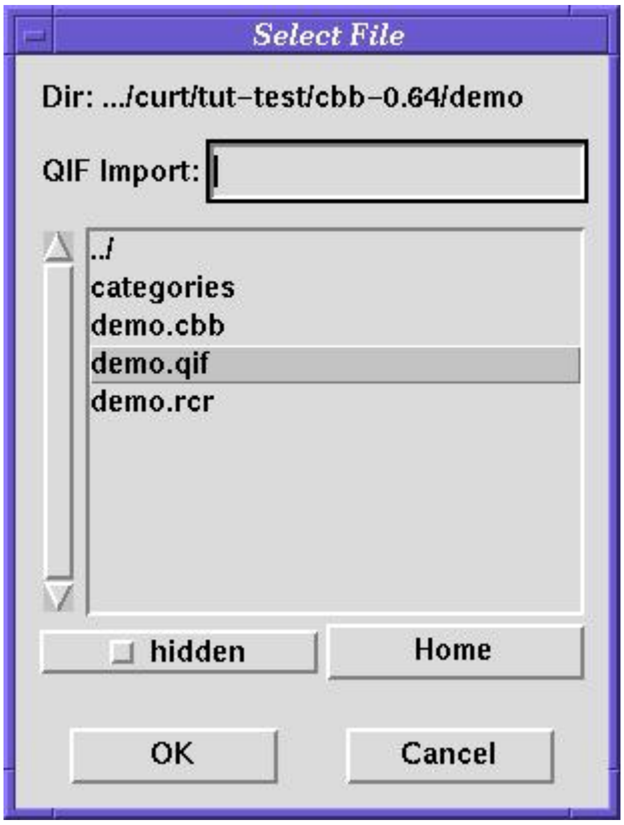


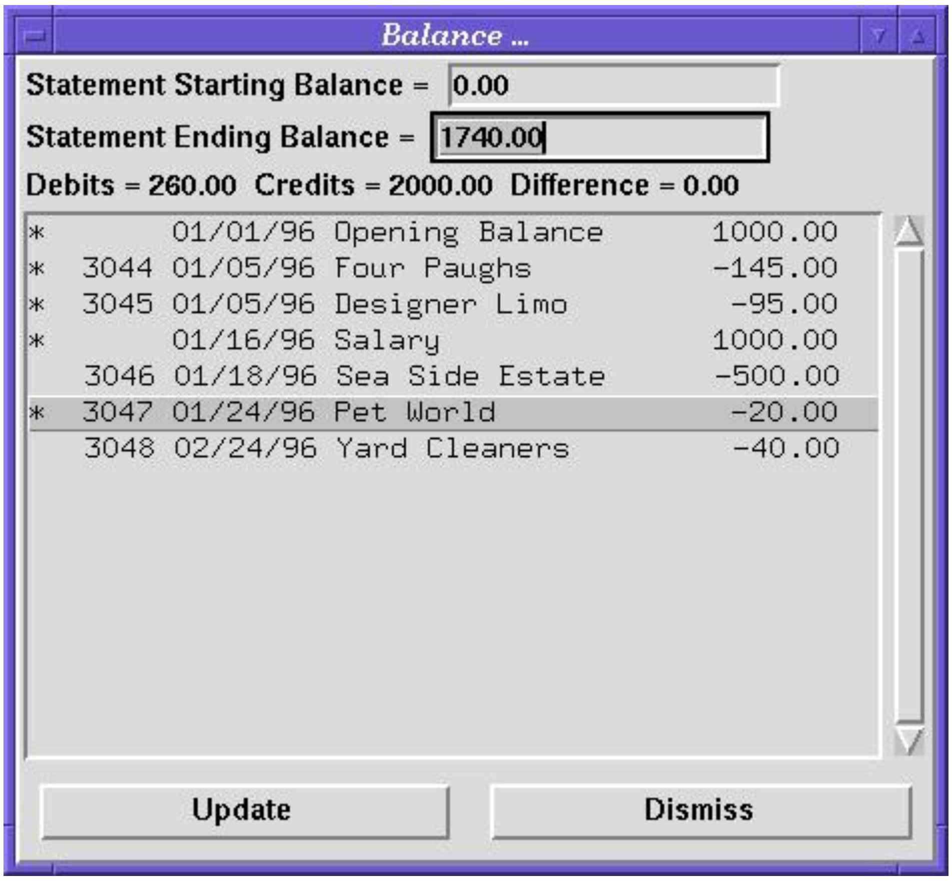
Figure 2. Choosing the Demo Import File

Now that you have some data to play with (Figure 3), try editing a transaction. Then try creating new transactions. Try playing around with “splits” to specify more than one category for a transaction. Play around with CBB until you get the hang of it.

Balance the Account

Now let's pretend you just received your bank statement in the mail and you want to reconcile your new CBB account. Let's also pretend that you didn't mess things up too badly while editing transactions...

- Find the **Balance** button near the bottom right-hand corner of the CBB window and click on it. This will bring up a list of all “uncleared” transactions.
- Enter a statement ending balance of $1740.00. (Leave the statement beginning balance as $0.00.)
- Select the first four transactions as well as the sixth transaction. (Pretend these were the ones that showed up on your statement.) As you select transactions, watch the “Debits = *n*, Credits = *m*, Difference = *i*” line.



Statement Starting Balance = 0.00
Statement Ending Balance = 1740.00
Debits = 260.00 Credits = 2000.00 Difference = 0.00

```
*       01/01/96 Opening Balance      1000.00
*  3044 01/05/96 Four Paughs          -145.00
*  3045 01/05/96 Designer Limo         -95.00
*       01/16/96 Salary               1000.00
   3046 01/18/96 Sea Side Estate      -500.00
*  3047 01/24/96 Pet World             -20.00
   3048 02/24/96 Yard Cleaners         -40.00
```

Figure 4. Reconciled Account

- When the *starting balance* - *withdrawals* + *deposits* = ending balance, click on the **Update** button to mark all the selected transactions as cleared.
- Congratulations: your account should be balanced.

When your next statement arrives and you run the balance routine again, you will only be presented with “uncleared” transactions to select. This is a good way to spot old checks that never were cashed, such as the mortgage payment that was “lost in the mail”. Note: By balancing your checkbook in this manner, two good things happen. First, when your brother finally cashes that check for $100.00—many months after you wrote it—your bank balance doesn't suddenly drop $100.00 from where you think it should be. Once that transaction was entered, the amount was subtracted from your bank balance. Second, it is easy to spot these situations—you can call up your brother and pester him to cash the check.

Reports and Graphs

Now that you have entered a bit of data, you may want to “understand” your data at a deeper level. CBB comes with several reports and graphs which can help you get a better idea of where and how your money is being spent. Feel free to generate and look at a few reports and graphs at this point.

Saving and Exiting

Wow! You've been slaving away for the last 10 minutes perfecting your demo account. Great job! That is about all there is to it. CBB isn't rocket science. It just boils down to “plus and minus”. Since this is not real data, you probably don't care to save it. However, if this had been an actual account, you most definitely would want to save your hard work.

If you want to save the changes you have made to an account, you must tell CBB to write the changes to a file. If you forget to save your work before you quit, CBB will remind you to do this. If you do something awful, like reboot your machine or log out without saving, you are not completely out of luck. CBB periodically saves a backup copy of your account with a file name such as **#account.cbb#**. Just rename this file to **account.cbb** and you will be back on track.

Additional Information

If you have made it this far, chances are you will be interested in some more information. I have a section devoted to CBB on my web page. Here you can find the complete CBB manual on-line, the readme file, the CBB FAQ, and several screen shots. You can even try running a live demo of CBB.

The URL for my home page is www.me.umn.edu/home/clolson/main.html.

It wouldn't be fair to end this article without mentioning two other free personal finance packages I am aware of. The first is Xfinans, another personal account manager. You can find out more about Xfinans at www.iesd.auc.dk/~lupus/xfinans.html. The second package is called Xinvest, a neat investment tracking application. Look for the source code at ftp://ftp.x.org/contrib/applications/.

Curt Olson works as a Unix systems administrator for the University of Minnesota, Department of Mechanical Engineering. He has been fascinated by computers from the time he first saw one:**? Please enter your name:** `Curtis Olson` *Hello Curtis. I am thinking of a number between 1 and 100...*“Ooohhh, I still get goose bumps thinking about how it remembered my name. Now, if only I could remember its name...”

Advanced search

Linux on Mobile Computers

Kenneth E. Harker

Issue #26, June 1996

On the road again? Whether business or pleasure takes you away from your desktop computer, Linux can come with you. Linux is a wonderful mobile computing environment, and reading this brief introduction to the state of Linux on laptop and notebook computers will help you get away from it all.

Just imagine the ability to pick up your Linux workstation and take it with you wherever you want to go. You can do it today. Linux can run smoothly on notebook hardware, taking full advantage of all the goodies that define mobile computing. Just as with desktop systems, however, there can also be challenges to overcome. Fortunately, extensive support for notebooks running Linux is becoming available as interest in seeing Linux on notebook computers continues to grow.

This article is not intended to be an in-depth aid to installation or configuration of Linux notebook systems. Such an article would be nearly impossible to write for desktop systems, and notebooks are no different. Rather, this article introduces both the present Linux desktop user and the Linux newcomer to the support that Linux offers for notebook hardware. Several on-line resources exist for more specific information beyond what is presented in this article, and they are mentioned below. The Linux Laptop Home Page collects pointers to all the information presented in this article and more, and is available at both http://www.cs.utexas.edu/users/kharker/linux-laptop/ (Figure 1) and http://www.redhat.com/linux-info/laptop/ (Figure 2).

Notebook computers can be quite different from desktop systems. One of the most important differences, when it comes to running Linux, is the process of integrating different hardware components. Desktop systems can be built by picking and choosing components that are known to work well with Linux. You can accessorize or replace unsupported hardware with the best cards and adaptors for your needs. Notebooks and laptops, however, are all-or-nothing packages. Aside from PCMCIA cards, there's very little you can do to customize your hardware configuration. There's also little you can do when it comes to controller chipsets or proprietary technologies—an unfortunate fact of life. Don't let this intimidate you, though; the vast majority of hardware on the market works quite well with Linux, and most of the rest works reasonably well with Linux.

Most notebooks have a similar feature set. At the time of this writing, only notebooks with Intel x86 family and compatible CPUs are capable of running Linux. The port to the PowerPC architecture may eventually support Linux. [Both the Linux/SPARC and Linux/Alpha ports have been run on laptops—ED]

Most notebooks feature keyboards with 85 keys, give or take a few, that are usually slightly smaller than those on desktop keyboards. Almost all notebooks come with PCMCIA drives capable of accepting two Type II or one Type III card. Most come with built-in 3.5" floppy drives and/or 5.25" CD-ROM drives. Displays vary widely in size, color depth, video memory, speed, and resolution. Many machines now come with integrated sound cards. Power management features are universal, but vary in implementation and effectiveness. Integrated pointer devices tend to have two buttons and come in three styles: trackballs, nub-shaped joysticks, and touchpads. Almost all notebooks feature a serial and parallel port, and offer some expandability or docking feature. Linux support exists for all these notebook features.

Installation of Linux on notebook computers should not be very much different from installing Linux on a desktop system. Most distributions let you choose from floppy, CD-ROM, hard disk, NFS, and even FTP installation sources. For the most part, choosing a suitable installation method for laptops is subject to the same criteria as for desktop systems.

There are a few exceptions, however. Some notebooks use external floppy disk drives, and there is really no standard interface for them. Most will allow you to boot and begin the installation process, but some do not support DMA transfer. Without DMA transfer capability, you may be able to boot and install from floppy, but the drive will not be useful for mounting other floppies.

Some distributions have boot disks that let you use a PCMCIA Ethernet card for NFS installation, assuming your card and PCMCIA controller are among those more commonly used. As with desktop systems, some early CD-ROM drives in notebooks used proprietary interfaces that are not supported by the Linux community. However, by carefully choosing industry-standard hardware, installing Linux on a notebook is no more or less of a task than it is on desktop systems.

PCMCIA

PCMCIA is an acronym that stands for Personal Computer Memory Card International Association. Since this is somewhat awkward to pronounce, PCMCIA cards are also frequently referred to as PC Cards. Physically, PC Cards are about the same width and height as credit cards and only a few millimeters thick. They provide one of the most versatile ways of customizing your notebook computer, and they are not limited to memory cards.

Readily-available PCMCIA cards implement RAM, small hard disks, SCSI adaptors, fax modems, Ethernet, sound, additional I/O ports, and more. Nearly all the cards on the market today are “Type II” cards. A few larger cards, such as those that implement hard disks, are “Type III” cards, and are about twice as thick. Most PC notebooks come with a PCMCIA drive that can handle two Type II cards or a single Type III card. To use a PCMCIA card, your machine needs a PCMCIA drive and some form of “card and socket services” software. The real trick to using PCMCIA cards in Linux lies in the software.

Most notebooks that come with MS-DOS and MS-Windows installed also include the vendor's card-and-socket-services software. This layer of software facilitates communication between the operating system and the drive itself—recognizing when cards are inserted or removed and determining which device driver is associated with which card. In the Linux universe, this software is called Card Services for Linux, and is actively maintained by David Hinds. It is included in most of the popular distributions, including Slackware, Red Hat, Caldera, Yggdrasil, and many others. It is also available at ftp://hyper.stanford.edu/pub/pcmcia/ as well as the Sunsite and tsx-11 Linux archives. The latest version at the time of this writing is 2.8.x. The PCMCIA HOWTO is available on the World Wide Web at hyper.stanford.edu/pub/pcmcia/doc/PCMCIA-HOWTO.html and can also be found on the Linux Documentation Project home page.

Card Services for Linux supports all but a very few custom PCMCIA drives, which are documented in the HOWTO. The support for individual cards is a little less complete. PCMCIA cards are no different from any desktop expansion card in the sense that the card needs a device driver to interface the hardware to the operating system. And like desktop expansion cards, not all PCMCIA cards have device drivers written for Linux.

Fortunately, essentially all PCMCIA fax modems and PCMCIA serial port adaptors operate on the same interface and are therefore supported. At the time of this writing, over three dozen brands of PCMCIA Ethernet cards are supported, as are numerous FLASH/SRAM cards and SCSI cards. You can find a list of supported cards at David Hinds' Linux PCMCIA Information Page at hyper.stanford.edu/~dhinds/pcmcia/pcmcia.html. David Hinds also maintains a mailing list for announcements about updates to the Card Services package and device drivers. Information on subscribing to this service is also available on this page.

The Card Services package also comes with all the current device drivers and several utilities. The most important of these is Cardinfo, a small X11 application that reports on the present state of your card sockets: what is in them, if they are active, what IRQ and I/O ports they occupy and which device (i.e. /dev/cua0) they are using. This is a very useful utility for finding and resolving IRQ or I/O port conflicts.

The latest versions of most Linux distributions include Card Services, but if your distribution doesn't include it, you can install it yourself. To do so, you will need to use kernel 1.2.8/1.3.30 or higher, compiled with support for loadable kernel modules. You must also have the entire Linux source tree to compile the package. Installing Card Services on an existing Linux system is explained in detail in the PCMCIA HOWTO and in the readmes that come in the package. This includes detailed information to help you identify and work around any IRQ or I/O port conflicts. When running, the core PCMCIA module takes up about 48K of RAM.

Card Services for Linux supports hot-swapping cards, and loads and unloads the proper card-specific device driver modules as you insert and remove PCMCIA cards. The Card Services module can also work with the Linux Advanced Power Management driver to help conserve battery life when your PCMCIA drive is inactive.

For those who are interested in hacking, David Hinds also provides some hints and suggestions for writing and debugging PCMCIA card device drivers in his HOWTO and in the PCMCIA Programmer's Guide.

Advanced Power Management

A major concern for many people taking their notebooks away from power outlets is how long their battery will last. Without any power management software, my 486DX4/75 notebook with a NiMH battery will last about forty-five minutes to an hour and a half away from a power outlet. While this will vary from machine to machine, it would be nice if this could be extended.

The goal of power management software is to manage the overall level of power consumption of the hardware by reducing or eliminating power consumption where it is not needed. This process is accomplished by communication between the hardware and operating system through a standard interface. For PC notebooks, this interface is known as the Advanced Power Management (APM) specification, version 1.1 at the time of this writing, and is defined by a document drafted by Microsoft and Intel. The specifications are intended to be operating system independent.

Although the APM specifications are intended to make power management independent of the operating system used, the unfortunate reality of the market is that some notebook manufacturers have decided to implement power management systems that work only with an MS-DOS or MS-Windows operating system.

Usually, new notebooks on the market will implement the specifications correctly, but there are many older models and even a few newer models that do not. If you happen to have one of those machines that does not implement APM correctly or sufficiently, you may be out of luck. Careful research of new machines can ensure that your machine is compatible.

APM works through communication between a properly designed system BIOS and an APM device driver in the operating system. BIOS stands for Basic Input/Output System, and is a Read Only Memory chip on your computer's motherboard. A system BIOS that implements APM can both read and modify the power consumption level of the hardware components in the machine. These components include your CPU, battery, screen, hard disk, floppy drive, PCMCIA drive, I/O ports, sound card, CD-ROM drive, and so forth.

The BIOS can communicate with the operating system's device driver, relaying this information so the operating system and BIOS can together make intelligent decisions about power levels. In this way, the operating system can power down or reduce power to those devices in the system that aren't in use, leaving more battery life for those devices that are in use. In addition, many notebooks feature a Suspend button that lets you manually put the notebook into a state of extremely low power consumption until you wake it up.

The Linux APM driver is maintained by Rik Faith, and the most recent version can be found already in kernel 1.3.46 or later. Support for APM can simply be chosen as a compile option in these kernels. Versions of the driver for older kernels and the 1.2.x kernels do exist as kernel patches, but are no longer supported by Rik and the other developers and lack some of the newer features. Those interested in APM support are therefore strongly encouraged to use a 1.3.x kernel. APM support will be a standard part of the next production kernel series, which will be called 1.4.*x* or 2.0.*x*.

In order for the APM driver to work, the system BIOS on your laptop's motherboard must support the APM version 1.0 or 1.1 interface, preferably version 1.1. It must also support 32-bit protected-mode connections. While most late model notebooks meet these requirements, a vendor's marketing claim of APM compliance is *not* sufficient. While the APM specifications strongly encourage laptop manufacturers to meet these standards, there is some leeway given to them. If APM support is important to you, make sure that the machine you are using meets these requirements.

In addition to the actual APM driver, there are several utilities available that use the APM kernel driver and the /proc/apm directory. These utilities are located at ftp://ftp.cs.unc.edu/pub/users/faith/linux and the current version of the utilities is available in the file apmd-2.1.tar.gz. This package includes a daemon process called apmd that logs battery status, and a utility named apm that simply outputs the information available in /proc/apm including the current battery level. For those running X-Windows, a simple utility called xapm displays a simple graph of battery life. And for those interested in hacking, a C library called libapm.a is provided so users can write their own utilities.

For those who have notebooks that do not have an adequate APM implementation in the BIOS, there is at least one other option available. A simple utility called hdparm is available which sets many IDE parameters, including how much inactivity an IDE drive should wait for before spinning down. The hdparm utility comes with every current Linux distribution, and can be obtained from ftp://tsx-11.mit.edu/pub/linux/sources/sbin/hdparm-2.7.tar.gz This is not true APM, but it's better than nothing. Hard disks use significant power and this simple utility can increase battery life up to 50%, in my experience.

X Window System

While some Linux users purchase commercial implementations of the X Window System, most people who use X-Windows with Linux choose the XFree86 implementation. XFree86 is a freely distributable implementation of the X Window System server for PCs running Unix and Unix-like operating systems. Odds are your favorite Linux distribution comes with the XFree86 X Window System server.

As anyone who has ever done it before can attest, configuring XFree86 can be quite a complex and occasionally frustrating task. Notebooks can complicate things further; while you can replace an unsupported video card in a desktop system with one known to work well with Linux, laptop owners do not have that option.

Notebooks come in all varieties and use all manner of components, including video chipsets. Some are well supported by the current XFree86 implementation, whereas others may not be supported or may offer only limited support. The Linux notebook community has both development and documentation efforts for the popular notebook video chipsets. Improvements to particular drivers can occur frequently. Acquiring up-to-date information about support for particular video chipsets is possibly the most important aspect of getting X-Windows up and running well on a notebook computer.

The most important source of information for running X-Windows on Linux notebooks is Darin Ernst's World Wide Web page X-Windows and Linux on Notebook Computers available at www.castle.net/X-notebook/index_linux.html (Figure 3). This site contains breaking news and links to numerous development efforts and their status. There are two World Wide Web pages that provide information on XFree86 support for the Chips and Technologies CT655xx series of video chipsets (the most widely used in recently-designed notebooks) as well as a mailing list for developers. Other prevalent video chipsets used in notebooks are produced by Cirrus Logic and Western Digital, and these are documented as well. Links to both these pages and many other sources of information are available from the X-Windows and Linux on Notebook Computers page.

In addition to the resources on the Web (See sidebar), several Usenet newsgroups are of interest to those wanting to run X on notebook computers. In particular, comp.os.linux.x and comp.windows.x.i386unix are the most relevant.

Sound, Networking, and Accessories

Many of the more recent notebooks come with integrated sound cards that are compatible with popular standards such as SoundBlaster. Many of these sound chipsets are new to the market or are reduced-size variations of chipsets used on desktop sound cards. Support for these chipsets can be found in the most recent release of the Linux sound driver. It is maintained by Hannu Savolainen at personal.eunet.fi/pp/voxware. The sound drivers are updated more frequently than most popular distributions, so checking the documentation at this site can produce pleasant surprises. But just as with desktop machines, SoundBlaster compatibility is often accomplished partly through hardware and partly through MS-DOS-based software. Therefore, you must take the same care when investigating a notebook's sound capabilities as when choosing a sound board for a desktop computer.

Networking with PCMCIA modems using SLIP or PPP is not substantially different from using a desktop machine with an internal or external modem. Simply remember to build SLIP or PPP support into the kernel you use. PCMCIA Ethernet networking is likewise similar to a desktop setup; you must have TCP/IP networking support compiled in your kernel and the appropriate PCMCIA Ethernet card device driver.

One exciting new networking project of which Linux has become a part is Mobile IP. This software supports transparent host mobility across TCP/IP networks and can be found at http://anchor.cs.binghamton.edu/~mobileip/ (Figure 4).

Almost all notebooks on the market today feature integrated pointer devices which vary in size and shape and in how they interface to the rest of the hardware. Most recently-designed notebooks have pointer devices that are PS/2 devices, so linking /dev/mouse to /dev/psaux and ensuring that support for PS/2 devices is in the kernel is all you need to do.

Some older notebooks used special controller chipsets for their pointer devices, some of which are supported in the kernel. If you have one of these older notebooks, your pointer device may or may not be supported. In any case, you can hook up a serial mouse to your serial port and use that.

Many notebooks allow the use of external keyboards or mice, frequently through an external PS/2 style port. Generally, these devices are managed through hardware and should work seamlessly with Linux. Most notebooks support external video monitors as well, and larger resolutions of external monitors can be exploited under X with properly configured XFree86 files.

Other Sources of Support

Linux has a strong tradition of user-based support such as the Linux Laptop Volunteer Support Database available at www.cs.utexas.edu/users/kharker/linux-laptop/volunteer.html. Volunteers provide their e-mail addresses and machine makes and models. Those looking for help can search for their hardware and find volunteers willing to answer installation or configuration questions. This can also be an excellent way to find a working XF86Config file. At the time of this writing there were over 200 volunteers from 36 countries.

The Linux Laptop Home Page that I maintain can serve as a good springboard to the world of documentation and support available for laptops running Linux and is available at http://www.cs.utexas.edu/users/kharker/linux-laptop/ and www.redhat.com/llhp/. Everything mentioned in this article and more can be found from the Linux Laptop Home Page, which is updated regularly.

Conclusion

The world of mobile Linux is very exciting. The ability to take your workstation with you appeals to many people. Concerns that choosing Linux as an operating system might limit access to or benefits from the various features that make notebooks really useful are largely unfounded. Active development and documentation efforts are supporting all aspects of mobile computing under Linux. If you've been considering running Linux on a notebook or laptop computer, there's never been more interest or support than there is now.

Kenneth E. Harker (kharker@cs.utexas.edu) maintains the Linux Laptop Home Page. In his spare time, he is a graduate student of the computer sciences at the University of Texas at Austin, an amateur radio operator, and an avid fan of *Babylon 5*.

Archive Index Issue Table of Contents

Advanced search

Using SmartWare Plus to Build the Integrated Office

Phil Hughes

Issue #26, June 1996

In many cases, the right answer for an office is an integrated solution. SmartWare Plus from Angoss is one possible approach.

Linux has made great inroads into development environments, become a good X terminal, and been used in embedded applications. However, to seriously penetrate the office market, Linux needs to not look like Linux. That is, when the average office user sits down at his system, he needs to see a means for accomplishing a clerical task, not a Unix or Linux command prompt.

There are ways to address this problem with available tools. Caldera has developed a desktop and applications suite. Others, such as Dr. Greg Wettstein of the Roger Maris Cancer Center, have built their own solutions using languages such as Perl and Tcl/Tk. Here at SSC we use the Progress database (now running on Linux) and then use external Linux tools (such as vi and groff) to do the support functions. Angoss offers another alternative with SmartWare Plus.

SmartWare Plus is an integrated office environment that includes a database, spreadsheet, word processor, and more. Also included is a Rapid Application Development System that allows you to write custom applications that use the features of the various SmartWare packages in an integrated fashion.

SmartWare has been around for quite a while and boasts over half a million users worldwide and over 600 software developers. SmartWare is available for MS-DOS and Xenix and has been very popular for use in local governments.

To get an idea of what you might do with SmartWare, think about an average office—say a dentist's office. You would need a system that allowed scheduling of patients, ordering supplies, patient billing, time management and letter writing.

You could train the receptionist (who probably also does billing, orders supplies, and probably most everything else in a small office) to use a word processor, a spreadsheet, an appointment manager and a database—but this would mean s/he would have to learn how to use all these programs, as well as how to use the operating system to load the different programs and move information between them.

One solution would be to purchase an integrated package, but you are in trouble if this package doesn't do exactly what you need it to right out of the box. A vendor of mass-market software is not going to make the changes you need or give you the source code so you can do it yourself.

With SmartWare, you can write the necessary spreadsheets, develop the scheduling and accounting software using the database, and then, using the Rapid Application Development (RAD) system, tie it all together so everything can be accessed from a single menu: Figure 1..

SmartWare also includes some utilities such as file copy, erase, and print that insulate the user from the operating system. While I don't get excited about a menu option to copy a file Figure 2. , menus do allow the user to accomplish tasks without even knowing what operating system is under the hood.

The advantages of using the SmartWare Plus package are:

- The same applications work under X-Windows and in character mode.
- Applications are portable between platforms/operating systems.
- All the necessary packages are integrated into one consistent package.

But, you pay a price for this. The disadvantages I see are:

- Not free.
- No source code for the system itself.
- The SmartWare user interface, while consistent among the different pieces of the package, is not consistent with other Linux/X applications.

Another important consideration is data security. Internal concerns need to be considered, such as access to payroll records, as well as mandated protection of confidential client records. Within the RAD system you can limit access to any application and thus any data.

Looking under the hood there is much more to SmartWare Plus than simply offering a GUI-based development environment. Also included is the SmartWare Programming Language—a complete programming environment containing standard structured programming constructs, over 200 commands and 300 built-in functions, multi-dimensional arrays and lots more. Thus, much can be done directly using the GUI-development tools but there is something under the hood if you need to do some serious programming.

Installation

When I first looked at this package I had a few misgivings concerning the installation procedure. The people at Angoss addressed them, but to be sure that the installation made sense, I asked Carlie Fairchild to give it a try. This is the result.

I then handed a copy to Bryan Phillippe, a systems technician at Zebu, a company related to SSC that produces Linux-based firewall systems. Bryan said, "SmartWare comes as a dynamically linked ZMAGIC a.out binary, tarred onto floppies. It installs easily for anyone with Unix experience, although the installation instructions themselves had some mistakes that might prevent a newbie from being able to install by following the instructions verbatim."

Once he had the product installed, Bryan started doing some development. The "Working with SmartWare Plus" Sidebar contains the remainder of Bryan's comments about the system. Figures 3 and 4 show the database screens he has developed so far. Figures 5 and 6 show the word processor and its spell-check mode, respectively.

Conclusion

There are two primary markets for SmartWare Plus. The first and most obvious market is to offer Linux as a new platform for existing applications. For example, people in a dentist's office already using SmartWare but who now need a multi-user answer are the perfect candidates for a move to Linux.

Secondly, SmartWare is a good fit for office situations where the computer is a tool to perform a particular set of well-defined tasks. For example, an order-entry system could be quickly built that would meet the specific needs of a company. New features (such as letter writing from the order system) could easily be added when needed.

As these two markets have identified millions of possible places where Angoss SmartWare Plus could be a good solution, I think I will stop there. Having done requirements statements for computer solutions for government and industry, and then searched for the right solution for hundreds of systems, I want to emphasize that it is crucial to find the right tool for the job. And, for a lot of jobs, this package, along with Linux, can be the right tool.

An evaluation copy of SmartWare Plus is available over the Internet and is also included on the InfoMagic archive CD set. This copy has save and print disabled, but for $50, you can get the key to enable it.

Commercial licenses cost $685 for the development system and $299 for each additional user. Angoss Software may be reached by e-mail at sales@angoss.com, by phone at 416-593-1122 or by fax at 416-593-5077.

Phil Hughes is the Publisher of *Linux Journal*, and the in-house nag for *WEBsmith*. He owns more than one, but less than 10, vehicles and tries to walk to work three times a year.

Scheduled Activity: cron and at

John Raithel

Issue #26, June 1996

Your Linux computer can be made to do the right thing at the right time.

The Linux utilities **cron** and **at** are related commands. The cron utility allows you to schedule a repetitive task to take place at any regular interval desired, and the at command lets you specify a one-time action to take place at some desired time. You might use crontab, for example, to perform a backup each morning at 2 a.m., and use at to remind yourself of an appointment later in the day.

Using crontab

The word “crontab” is a Unixism for “chron table”, or time table. You create a table in the required format, specifying commands and times to execute the commands. Commands you put in the table can be any executable programs—for example, a command in /usr/bin or a shell script you wrote. You use the crontab command to create, edit, or list the table, and the system cron daemon reads the table and executes the commands at the times specified.

The cron Daemon

The cron daemon is normally executed at system startup and does not exit. On my Linux system, the cron daemon is actually Matthew Dillon's **crond**, and is started in /etc/rc.d/rc.M with the following line:

```
/usr/sbin/crond -l10 >/var/adm/cron 2>&1
```

On some Linux systems, Paul Vixie's cron daemon is used, in which case the name of the daemon is simply **cron**. Also, on systems with newer versions of init, cron is started from the /etc/init.d/cron script.

You can check to see if a cron daemon is running on your system with a command such as the following:

```
$ ps -ax | grep cron
raithel  733  pp0 S   0:00 grep cron
root      25   ?  S   0:00 /usr/sbin/crond -l10
```

In this case, we see that the **crond** daemon is indeed running.

The crontab Table

When the cron daemon starts, it reads the various crontab tables in the crontab directory, normally /usr/spool/cron/crontabs. To create or change your crontab file, use crontab's **-e** option:

```
$ crontab -e
```

You are placed in a text editor with a copy of your current crontab file, if it exists, or a blank file, if it does not. The text editor you get is determined by the setting of your **VISUAL** environment variable (or **EDITOR**, if **VISUAL** is not set) and is usually the **vi** editor, if you have not specified otherwise.

To schedule commands with crontab, you must use the format cron recognizes in a crontab file. The format is not exactly mnemonic, so I create a crontab file with a header commented out that provides the necessary information:

```
# minute (0-59),
#    hour (0-23),
#       day of the month (1-31),
#          month of the year (1-12),
#             day of the week (0-6, 0=Sunday),
#                command
```

Each crontab entry is a single line composed of these six fields, separated by white space. Specify the minute a command is to be executed with the digits 0 through 59 in the first field, the hour with 0 through 23 in the second field, the day of the month with 1 through 31 in the third field, the month of the year with 1 through 12 in the fourth field, and the day of the week with 0 through 6 in the fifth field. Place the command to be executed in the sixth field.

At first glance it may appear that redundant or conflicting information is required because there are two “day” fields—day of the month and day of the week, but this is really just to permit different scheduling algorithms. For instance, you may want to be reminded to attend a meeting every Tuesday or to pick up your paycheck every 15th of the month. Enter an asterisk (*) in the day field you are not using. You can use both day fields if you prefer to have the command execute on, say, the fifteenth of the month **as well as** every Tuesday.

Ranges are specified with a dash. If you want to specify the eighth through the fifteenth days of the month, enter **8-15** in the third field. Non-consecutive entries in a field are separated by commas, so **1,15** in the third field means the first and fifteenth of the month. To specify all values for a field, for example every month of the year, enter an asterisk (*****) in the field. (Note that to specify every day you must enter ***** in **both** day fields.)

Here is an example crontab file with two entries:

```
# minute (0-59),
#   hour (0-23),
#      day of the month (1-31),
#            month of the year (1-12),
#               day of the week (0-6, 0=Sunday)
#                  command
12  4  *     *  *  /usr/local/bin/backup
5   3  10-15 4  *  echo "taxes due" | mail jones
```

The first line after the comments causes a backup script to execute early each morning at 4:12 a.m., and the second line causes the user jones to get a mail message for six days in April as a reminder that taxes are due. In general, it's a good idea to execute crontab commands at off hours like these to reduce any effect on system load during normal usage hours.

If you don't specifically redirect standard error and standard output, they are mailed to you as owner of the crontab file when the command executes. In the example above, if the user jones cannot be found, you would be mailed the output as well as an error message.

After editing the crontab file, save it and exit from the editor. A file is created for you in the crontab directory. For example, the crontab for root is the file /usr/spool/cron/crontabs/root. This file is read by the system cron daemon and stored in an internal format, where it will remain to be periodically executed until it is changed or deleted.

To view your current crontab file, use the **-l** (for “list”) option:

```
$ crontab -l
```

To delete your file, use:

```
$ crontab -d
```

If you are superuser, you can delete any user's crontab file with:

```
# crontab -d username
```

where ***`username`*** is the user's login name.

The crontab commands discussed above work fine on my Linux system and should work on System V and BSD Unix systems, as well. One thing to be aware of when using crontab on other systems or moving crontab files to other systems is that some cron daemons allow the superuser to restrict crontab service by the creation of cron.allow and cron.deny files. Refer to the specific system documentation for details.

Also, most versions of cron provide an /etc/crontab file which has an extra field in it—the user as which to execute the command. Again, check the documentation for your version of cron for more details.

Using at

Use **at** when you want to execute a command or multiple commands once at some future time.

In Linux, the at command requires that the atrun command be started in root's crontab file. Many Linux distributions ship with at enabled, but some do not. To enable the at utility on your system, become superuser and edit root's crontab file:

```
$ su root
Password:
# crontab -e
```

and add the following line:

```
* * * * * directory/atrun
```

where ***directory*** is the location where the atrun executable is stored. On my system that's /usr/lib, so the entry is:

```
* * * * * /usr/lib/atrun
```

This causes atrun to be executed every minute. After a minute or so of adding the atrun line and saving the crontab file, any existing at commands are evaluated and executed if the time is right. (Before this, you may have been able to enter at commands, but they were never executed.)

To demonstrate the at command, let's have it print “hello” on your current terminal window after a few minutes. First, get the time and your current terminal device:

```
$ date
Tue Oct  3 15:33:37 PDT 1995
$ tty
/dev/ttyp2
```

Now run the at command. Specify a time in the command line, press Return, and then enter the command, followed by another Return and a Ctrl-D:

```
$ at 15:35
echo "hello" > /dev/ttyp2
```

```
^D
Job c00ceb20b.00 will be executed using /bin/sh
```

The at command takes input up to the end-of-file character (generated by pressing ctrl-D while at the beginning of a line.) It reports the job number and informs you that it will use /bin/sh to execute the command. In two minutes, **hello** should appear on the display of /dev/ttyp2. Note that you can enter a series of commands, one per line—at will read each line up to the end-of-file and execute the file as a /bin/sh shell script at the specified time.

Suppose you want to set an alarm. One way to tell at to do something is to use the relative form of timing, specifying a time relative to **now**. If you want your computer to beep at you in 25 minutes, enter:

```
$ at now + 25 minutes
echo ^G > /dev/ttyp4
^D
Job c00ceb7fb.00 will be executed using /bin/sh
```

and you are beeped in 25 minutes. There is a great deal of flexibility allowed in entering time specifications. For example, at recognizes military time, “am” and “pm”, month abbreviations, time notation that includes the year, and so on. My at man page even claims at accepts **teatime**, **noon**, and other constructs. Refer to the at man page for more examples of valid time specifications.

You must tell at your tty location, or it won't send output to your terminal window. If you prefer, you can receive mail:

```
$ at 4:55pm Friday
echo '5 p.m. meeting with Carol' | mail raithel
^D
Job c00ceb7fb.01 will be executed using /bin/sh
```

To get a list of your pending at jobs, enter:

```
$ atq
```

If you are superuser, atq shows you the pending at jobs of all users. To delete a job, enter:

```
$ atrm job_number
```

where ***job_number*** is the job number returned by atq. The superuser can also remove other user's jobs.

A Reminder Script Using at

The following is a simple script that makes it easier for me to use at to send myself reminders. The script sends mail to the user containing the message line(s) entered at the prompt at the time specified. It also displays some syntax examples of how to specify time, which I find a useful memory refresher.

Notice the script, as written, requires you to have a Msgs directory in your home directory. I created $HOME/Msgs, rather than using something like /usr/tmp, so the messages are more private until they are deleted by the script.

```
#!/bin/sh
echo "Enter your reminder message.
When finished, enter a period (.) at
the beginning of a line and press Enter.
(Or press Ctrl-C or DEL to exit.)"
while :
do
    read MESSAGE
    if [ "$MESSAGE" = "." ]
    then
        break
    else
        echo $MESSAGE > $HOME/Msgs/message.$$
    fi
done
cat << !!
Enter time and day you want to receive
the message, for example:
      0815am Jan 24
      8:15am Jan 24
      now + 1 day
      5 pm Friday
Then press Enter.
!!
read TIME
echo \
  "at $TIME mail $LOGNAME $HOME/Msgs/message.$$"
at $TIME  << !!
mail $LOGNAME < $HOME/Msgs/message.$$
rm -f $HOME/Msgs/message.$$
!!
exit 0
```

Some Final Thoughts

The user's interface to the crontab and at commands is very similar across different versions of Unix, but implementations of underlying directory structures, daemons, and access controls may differ. Be sure to review your system documentation to take advantage of all aspects of these powerful commands.

There's practically no limit to the use of crontab and at, but let me offer a few words of warning. First, consider security issues when enabling user crontab and at permissions. Obviously, a disgruntled co-worker could leave a “time-bomb” of some sort, limited only by his other permission restrictions. Many versions of crontab and at allow you to specify “allow” and “deny” files to control which users have access to the utilities. You could also use root's crontab to check for and remove user crontabs (or any other files) if you want to.

Also, debug your crontab file entries thoroughly. Check that they are working. These entries are usually scheduled to execute at times of low-usage, so it is unlikely you'll be around to observe them at the time.

John Raithel is a consulting technical writer specializing in documenting the system and network administration of the Unix operating system, and is currently working on World Wide Web and firewall documentation for Silicon Graphics, Inc. He lives in a small town on the central California coast where he plays with his Linux and SunOS “mini-network”. He can be reached via e-mail at raithel@rahul.net.

Archive Index Issue Table of Contents

Advanced search

Uniforum '96

Belinda Frazier

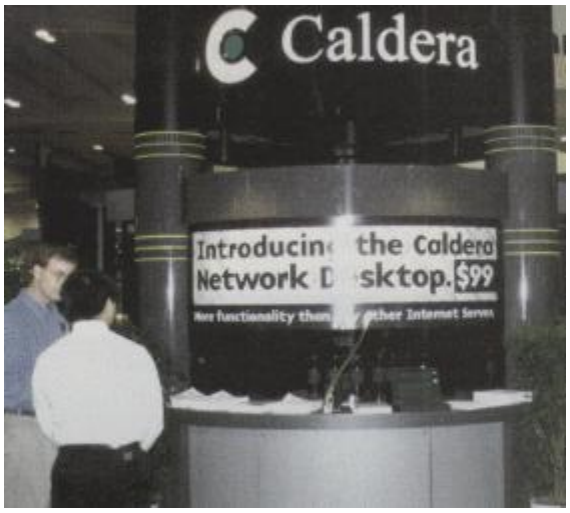
Issue #26, June 1996

UniForum '96, now billed as a Unix and Open Systems show, attracted about 20,000 attendees in San Francisco in February.

UniForum '96, now billed as a Unix and Open Systems show, attracted about 20,000 attendees in San Francisco in February. It included 90 tutorials, seminars, and keynotes from Lew Platt, of Hewlett-Packard, and Scott McNealy of Sun MicroSystems.

Left to right: Adam Richter of Yggdrasil Computing, David Fickes of Prime Time Freeware and Bob Young of Red Hat Software, standing in front of the WGS Booth, with Mark Bolzern in the background.

UniForum was founded in 1981 to advance the use and standardization of Unix and open systems.



Caldera demonstrated the Caldera Network Desktop at Uniforum '96.

Besides offering a Unix-focused education forum, UniForum provided an opportunity for companies to exhibit their products. One section of the exhibition floor was labelled “Linux Area” and included the booths of Linux International, WorkGroup Solutions, SSC, and Enhanced Software Technologies. Other vendors with Linux products, such as Caldera and Comtrol, were not far away.

John “maddogg” Hall at the Digital Equipment Corporation Booth.

An Introduction to Rlab: A High Level Language for Scientific and Engineering Applications

Ian Searle

Issue #26, June 1996

Rlab stands for “our lab”. It is available to almost everyone who needs a computational tool for scientific and engineering applications, because it is freely available, and it runs on many platforms.

I started working with high level languages when I realized I was spending far too much time writing Fortran and C language programs as a part of engineering analyses, test data reduction, and continuing education. I searched for a better engineering programming tool; except for Matlab, and some Matlab-like programs (all commercial), I came up empty-handed (this was in 1989). I did not feel Matlab's language design was powerful enough, but I would have used it had it been lower in cost and available on more platforms. I diverted my “off-hour” studies to interpreter construction, and started prototyping Rlab. Within a year, I released version 0.10 to the Internet for comment. Now, almost five years later, Rlab has undergone significant changes and improvements primarily due to excellent assistance from users around the world.

Rlab does not try to be a Matlab clone. Instead, it borrows what I believe are the best features of the Matlab language and provides improved language syntax and semantics. The syntax has been improved to allow users more expression and reduce ambiguities. The variable scoping rules have been improved to facilitate creation of larger programs and program libraries. A heterogeneous associative array has been added to allow users to create and operate on arbitrary data structures.

Overview

The term “high level language” has many different connotations. In the context of Rlab it means that variables are not explicitly typed or dimensioned. A variable's type and dimension are inferred from usage, and can be arbitrarily changed during a program's execution. This automatic typing and dynamic allocation free the scientific programmer from many of the more time consuming programming duties normally associated with languages like C or Fortran.

In Rlab's case, high level also means interactive. Since one of the purposes is to provide convenient means for data manipulation and visualization, the program can be run in either batch or interactive mode. Typing **rlab** at the shell prompt starts the program. Rlab gives the user the command line prompt prompt **>**, from whence program statements are issued and executed.

What do you get? What does it cost? Most people find that they can develop a working program far faster than with C or Fortran. Furthermore, the built-in graphics capabilities and interactive mode of operation promote experimentation and exploration of data and methods. The price you pay is slower program execution in *some* instances. Performance tests between compiled and interpreted languages usually show the compiled language to be faster in all respects. If your Rlab program uses scalar operations exclusively, and uses none of the built-in linear algebra functions, then it will be substantially slower than the Fortran or C version. However, if you take advantage of Rlab's optimized array operations, built-in linear algebra and utility functions, you may find that Rlab will perform quite well.

Although I cannot provide a complete introduction in a single article, I can show you enough of the language to help you decide if it is something that you would want to try. A more complete introduction is provided in “The Rlab Primer”, which is included in the source and binary distributions.

Data

The numeric array, or matrix, is the most commonly used data type. The numeric array can be one-dimensional (a vector) or two-dimensional. Most often, matrices are created by reading data from another program or from a file. For simpler examples, we will create our matrices interactively. Array/ matrix syntax uses square brackets, **[ ]**, like the C language, to delimit the matrix, and semi-colons to separate each row from the next. So, to create a matrix within Rlab, the user types:

```
 > a = [1, 2; 3, 4]
```

Which provides the following response:

```
    a =
          1          2
          3          4
```

Matrix referencing and assignment uses the following syntax:

```
> a[1;] = 2 * a[2;2,1]
 a =
        8          6
        3          4
```

The semi-colon delimits row and column specifications. The absence of a row or column specification selects all of the rows or columns. Additionally, the user can specify particular indices, or combinations of indices, in any order. In the previous example, **a[2;2,1]** selects the second row of **a**, and the second and first columns (in that order) of that row. Therefore, **a[2;2,1]** evaluates to **[4,3]**. This quantity is then multiplied by 2, and the result is assigned to the first row of **a**.

In this article real numbers will be used in all of the examples. However, Rlab can deal with complex numbers equally well. To enter a complex number, the imaginary constant **i** or **j** is used (don't worry, you can still use **i** and **j** as variables). For example: **a = 1 + 2j** creates the complex number with real part 1 and imaginary part 2. All numeric operators and functions (where they make sense) work with complex values as well as real values.

The traditional numeric operators are overloaded. When working with scalar operands, the results are what most people would expect. However, when working with array/matrix operands, the numeric operators behave differently. Addition and subtraction function in an element-by-element sense between the two operands. The multiplication operator ***** performs the matrix inner-product. To illustrate:

```
> a = [ 1 , 2 , 3 ];
> b = [ 4 , 5 , 6 ];
> a + b
        5          7          9
> a' * b
        4          5          6
        8         10         12
       12         15         18
> a * b'
       32
```

Note (in the first two lines) that a **;** at the end of a statement suppresses printing the result. The division operator **/** performs traditional division on scalar operands, and solves sets of simultaneous linear equations with matrix/vector operands. A set of element-wise operators exist to perform element-by-element operations between arrays. The element-wise operators are two-symbol operators; the first symbol is always a **.** so the element-wise division operator is **./** and the element-wise multiplication operator is **.***:

```
> a = [1,2,3;4,5,6];
> b = 1./a
 b =
        1        0.5      0.333
     0.25        0.2      0.167
> a.*b
```

```
           1           1           1
           1           1           1
```

In addition to the numeric class of data, there is a string class. String arrays/ matrices are handled in exactly the same way as numeric arrays:

```
> s = ["how", "to create" ;
>      "a string", "within rlab"]
 s =
how          to create
a string     within rlab
```

As you can see, string arrays consist of arbitrary length strings. There are no restrictions upon the relative size of strings in an array. Strings can be concatenated with the **+** operator. The other (traditionally numeric) operators do not operate on string arrays.

Lists

One of the benefits of the C language is the ability to create data structures to suit a particular programming task. Rlab offers a similar programming construct called a list. In their simplest form, lists are single-dimension associative arrays. However, since list elements can be strings, numbers, arrays, functions, and other lists, they provide a powerful multi-dimensional tool. Lists are created in a manner similar to numeric arrays:

```
> l = << 3*pi ; rand(3,4) ;
>        type = "my-list" >>
 l =
   1            2            type
```

The **<< >>** delimiters contain the list elements, with **;** separating the elements. The **=** is used to assign an index name (which cannot be numeric) to an element, as in the third element of the list above. If a list element is not assigned an index, then a numeric value is assigned by default, as in the first two elements.

Elements of a list are referenced by their index: **l.["type"]** returns the string **"my-list"**, and **l.[1]** returns the value of **3*pi**. In the case of explicit index names, shorthand notation can be used: **l.type** will also return the string **"my-list"**.

List indices can be replaced by expressions that evaluate to a string, or a numeric-scalar, allowing users access to selected elements in an automated fashion. If **index** is a variable containing the string **"type"**, the expression **l.[index]** will access the **type** element and return the string **"my-list"**. If **index** contains the number 1, **l.[index]** will return **9.42**. Lists will be discussed again later in the article.

Getting On With It

Rlab provides the user with conditional and if-statements for conditional execution, for and while-statements for looping capability, and functions or subroutines. These capabilities will be introduced as we proceed with some examples.

Rlab functions are a little unusual, and deserve some attention. Functions, while not first class, are objects, referred to with variables. Thus, to create and save a function, it must be assigned to a variable. The function argument passing and variable scoping rules were designed to facilitate creation of larger programs and program libraries. Our first example, integrating an ordinary differential equation will illustrate some of these features.

A popular example is Van der Pol's equation, because it is simple, non-linear, and it demonstrates limit-cycle oscillation. Rlab has a built-in integration engine, and several “rfile” integrators. An rfile is a file that contains an Rlab program or statements. The integration function needs as input: a function that returns the value of the derivative, values defining the start and end of the integration interval, and the initial conditions.

The entire problem could be performed interactively. However, I have chosen to put the program in a file (int.r). This allows me to edit and re-execute without a lot of repetitive typing. For this article, I have used the shell-escape feature (**** in the first column) to cat the file through pr to generate Listing 1.

Since the ode function integrates first order differential equations, we must write Van der Pol's equation:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0$$

as two first order equations:

$$\dot{x}_1 = x_1\mu(1 - x_2^2) - x_2$$

$$\dot{x}_2 = x_1$$

The function, which calculates and returns,

$$\ddot{x}_i$$

is written and assigned to the variable **vdpol**. After the function is defined, variables **t0**, the integration start time; **tf**, the integration final time; and **x0**, the initial conditions, are initialized (lines 9-11). **tic** and **toc**, builtin timing functions, are used around the call to **ode** to tell us how long the integration takes.

Once this file is finished, the following command will execute it, as if each line was entered from the prompt.

```
"> rfile int
ODE time:      0.620
```

This simple problem runs fairly fast. The output from ode, a matrix of the time-dependent values of,

x_i
is stored in the variable **out** (line 14). Convenient data visualization is a real plus when investigating the behavior of differential equations. In this instance we would like to look at

x
(**x**) and

$\dot{x}$
(**xd**) versus time. We would also like to look at the phase-plane for this problem, which is a plot of

x
versus.

$\dot{x}$
We can do this with the plot function, which plots matrix columns. If the input to plot is a single-column matrix, the matrix row indices are plotted against the abscissa-axis, and the matrix column elements are plotted against the ordinate-axis. If the input to plot is a N-column matrix, then the values in the first column are plotted against the abscissa-axis, and the remaining columns are each plotted against the abscissa-axis.

A plot, similar to Figure 1, can be created with the command **plot(out);**. Since the first column of **out** is time, and the second and third columns are **x** and **xd**, all we have to do is give the unaltered matrix to plot. If we want to plot the phase-plane, as in Figure 2, we need to specify that we want the third column plotted against the second. To do this simply extract the second and third columns from **out** like so: **plot(out[;2,3]);**

The Plplot graphics library provides the builtin 2 and 3D graphics capability for Rlab. Builtin functions for most common graphics capabilities such as: 2D, 3D, mesh-plots, contour-plots, and histograms are supplied. Plplot supports most common graphics output devices, most notably X-Windows and PostScript.

It is possible that Plplot graphics may not be sufficient. In these cases the ability to get properly formatted data to another program is paramount. There are several methods of interfacing Rlab with other programs. Like most good Unix applications, Rlab will read from stdin and write to stdout. There are functions for writing matrices in ASCII format, as well as a C-like [f]printf function. There is also a system function that allows Rlab programs to do anything that can be done with the Unix shell. However, what makes interfacing with other programs easiest is the facility for writing to, and reading from a process (pipes).

Input/Output

The user-end of the I/O system was designed to be as simplistic as possible, without restricting capability for those who need it. Files and processes are identified with strings. Each function capable of reading or writing will open and close files as necessary. However, **open** and **close** functions are provided for special circumstances. There are several variations of read and write functions that understand all of Rlab's internal data structures, and offer some degree of Matlab file compatibility. The **getline** and **strsplt** functions used for special ASCII input needs, a **fread** used for binary input, and a **fprintf** used for specially formatted output round out the I/O functions.

As you might expect, the strings **"stdin"**, **"stdout"**, and **"stderr"** point to their Unix system counterparts. Any function that performs file I/O can also perform process I/O through pipes, by simply replacing the filename string with a process-name string. A process-name string is a string that has a **|** as its first character. The rest of the string is any shell command. Rlab will create a pipe, forking and invoking the shell. The direction of the pipe is inferred from usage. This facility makes getting data to and from other programs rather simple. A good example is the Rlab-Gnuplot interface, which is written entirely in the Rlab language, using the process I/O capability to get data and commands to Gnuplot.

As a demonstration, we will explore process-I/O with a simple interface to the X-Geomview program. X-Geomview is a powerful 3-dimensional graphics engine, with an interactive GUI. X-Geomview can read data from files, and it can also read data/commands from stdin. X-Geomview uses Motif, but statically linked Linux binaries are available (in addition to the sources) from www.geom.umn.edu/software/geomview/docs/geomview.html.

In this example I will generate the data for, and plot, the classic sombrero. The code is listed in Listing 2.

The data for the example is completed by line 14; from there on, we are simply sending the data to the X-Geomview process. The variable **GM** holds a string, whose first character is a **|**, indicating a process to the remainder of the string should be opened. The following statements (lines 16-21) send object definition to X-Geomview, and lines 23-30 include the nested for-loops that send the polygon vertex coordinates to X-Geomview. A snapshot of the X-Geomview window containing the result is presented in Figure 3. Of course, a much better way to make this type of plot is to create a function that automates the X-Geomview interface (this will be included in the next release of Rlab).

Manipulating the Workspace

High level languages are great for prototyping a procedure, but often fall just short of useful when it comes time to use the program in a “production” situation. In this example we will pretend that we have just developed a method for computing time-frequency distributions. Actually we are going to use Rene van der Heiden's tfd function, which is derived from the information in Choi and Williams' paper, “Improved Time Frequency Representation of Multicomponent Signals Using Exponential Kernels”.

Now we want to use tfd to process a large amount of data. Since tfd executes reasonably fast, we would hate to have to re-code it in some other language just to be able to handle a large amount of data. Suppose you have many files of time-history data that you wish to “push” through tfd. Some of the files contain a single matrix of event data, while others contain several matrices of data. You would like to be able to write a program that could process all such files with a minimum of user intervention. The difficulty for some languages is the inability to manipulate variable names, and isolate data.

Rlab addresses this problem with lists. Lists allow the creation, and isolation of arbitrary data structures, and provide a mechanism for systematically manipulating workspace variables, and variable names. I showed earlier how list elements could be accessed with strings. Lists can also be indexed with a string variable, or for that matter, any expression that evaluates to a string.

The interesting thing I have not disclosed yet is that the entire workspace can be treated as a list! Access to the workspace is granted through the special symbol **$$**. You can use **$$** as the name of a list-variable. For example, you could use the cosine function like: **$$.["cos"](pi)**, or: **$$.cos(pi)**. The first method offers the most flexibility. Now that we know about this special feature, we can handle our problem with relative ease. The program will read each file that contains data (they match the pattern *.dat) one at a time, compute the time-frequency distribution, and save the distribution for each set of data in the workspace. When processing is complete, the new workspace will be saved in a single file for later use.

The program just described is contained in Listing 3. There are several things I should point out:

- Line 1: The require statement statuses the workspace for the named function. If the function is found, nothing happens. If the function is not found, it is loaded from disk.
- Line 10: The getline function reads ASCII text files, and automatically parses each line into fields (sort of like awk). The fields (either strings or

numbers) are returned in a list. When getline returns an empty list (as detected by the length function), the while-loop terminates.

- Line 12: Each filename is stored in a string array.
- Line 13: The readb function reads all the data from each file, and stores it in the list, **$$.[filenm[i]]**. This is a variable in the workspace that has the same name as the filename. For instance, if the first file is “x1.dat”, then a list-variable will be created called “x1.dat”.
- Line 24: Now we are going to operate on the data we have read. The program will loop over the strings in the array **filenm**.
- Line 26: For each file (i), the program will loop over all the data that was in each file. The members function returns a string array of a list's element names.
- Line 28: This is it! **$$.[i]** is a list in the workspace (one for each data file). **$$.[i].[j]** is the jth variable in the ith list. So we are computing the time-frequency distribution for every matrix in every file we have read. **$$.[i].[j+"y"]** creates a new variable (same as the matrix name, but with a “y” tacked on the end) in each list for each time-frequency distribution that is performed.

Wrap-Up

There are many features, functions, and user-contributed programs that I have not discussed. Of special note are the extensive set of linear algebra functions. Rlab provides a very convenient interface to the LAPACK, FFTPACK, and RANLIB libraries available from Netlib and Statlib.

By now you have should have seen enough of the language to decide whether it is worth the effort to try it out. If you would like to find out more about it you can check out: www.eskimo.com/~ians/rlab.html. The Linux binaries of Rlab are now being distributed in RPM format, and are available at ftp://ftp.eskimo.com/u/i/ians/rlab/linux. Now that ELF distributions are available, and out of beta testing, and considering how much better dynamic linking is with ELF, Rlab binaries are built using the ELF object file format.

Acknowledgments

The Plplot graphics library is provided by Dr. Maurice LeBrun at the University of Texas. The underlying linear algebra subroutines (LAPACK, and the BLAS) are from the Netlib repository. And, of course, none of this would have been possible without GNU tools. There have been many other contributions to Rlab by various individuals over the years. The ACKNOWLEDGMENT file in the source distribution tries to mention everybody.

Where to Get Rlab

Rlab is available with over a hundred rfiles of various sorts, many contributed by users. Professor J. Layton, at Clarkson University, Potsdam NY, is in the process of finishing up the Rlab Controls Toolbox. A port of Professor Higham's (University of Manchester/UMIST) Test Matrix Toolbox is also available. For more information go to www.eskimo.com/~ians/rlab.html

Ian Searle currently works in the aerospace research field in Seattle Washington, and works on Rlab in his spare time.

Letters to the Editor

Various

Issue #26, June 1996

Readers sound off.

GNU Restrictions?

I have been a fan of Linux for some time, and lately also of *LJ*, which I consider an excellent source of information. I have, however, been reluctant to address Linux as a *target* platform, because of the restrictions imposed by the GNU General Public License.

If I understand correctly, I may not compile a program with gcc under Linux and then expect to market it without accompanying source code. Also, I may not deny my licensee the right to re-distribute the program, or even sell it. This is because my application would constitute a “work based on gcc”, as defined in paragraph 1 of the GPL, and also because it would contain library code covered by the GPL.

But then, browsing through your magazine I found out that, for example, Caldera imposes much more restrictive terms on its products. Also, I have seen an ad about Mathematica for Linux, and I doubt that Wolfram Research is willing to qualify its product as a “work based on gcc”.

Clearly I am missing something. The question is, how can you market a commercial product under Linux and make sure that your customer is not re-selling it, or maybe installing it on 600 machines? Do you have to use a compiler other than gcc (is there any)?

I appreciate any advice you may give on the subject. Keep up the excellent work.

—Luca Cotta Ramusino lcotta@systemy.it

Common Misconception

First of all, compiling with gcc does not make your application a “work based on gcc”. Second, the C library is not covered by the GPL, but by the LGPL, the GNU Library General Public License, which allows you to distribute applications linked to shared libraries without inheriting copyright restrictions. Third, there are at least two other C compilers available for Linux; Linux FT comes with a different compiler as the default system compiler, and lcc is also available.

So you can safely target Linux with your current GNU toolset.

New Swatch Location

Greetings. I read the January and February issues of *LJ* with great interest, especially the security section. In the February issue, you have the site for swatch as being sierra.stanford.edu:/pub/sources. It has moved to ftp.stanford.edu:/general/security-tools/swatch. I thought that this might be useful to anybody else who is looking for it...

Regards,

—Duncan Hill dhill@sunbeach.net

[The url he mentions has been corrected for this archive CD —Ed]

Unix Wars, Redux

Michael K. Johnson

Issue #26, June 1996

Linux is not Unix, but since Linux is a Unix “clone”, wars in the Unix world affect the Linux world.

For years, Unix vendors fought each other for market share by each implementing their own proprietary extensions to Unix, each trying to use their own extensions to sell their version of Unix—usually on their own hardware. No one blames them for competing, but the result was not good for the whole Unix marketplace. Many slightly different products, which had different names but somehow were all understood to be Unix, more or less, bewildered and bothered consumers.

Because they recognized the damage this situation could cause, Unix vendors have been making interesting noises for years about closer co-operation with each other. Novell bought Unix, then gave the Unix trademark to X/Open, to create an open branding process in order to help bring the Unix world together. X/Open then released the Single Unix Specification, to which all vendors' versions of Unix are required to conform in order to use the trademark. To an amazing extent, this has been a success; quite a few products that could not use the trademark before have now been “branded”.

Last fall, at Unix Expo, HP and SCO announced that they were working together to buy Unix development rights from Novell, and that they were going to develop the new standard 64-bit Unix. While several versions of Unix have been more or less 64-bit in the past, the Single Unix Specification (SUS) does not explicitly address 64-bit issues, and SCO and HP (especially HP) were going to supply the Unix world with 64-bit Unix.

This was expected to provide the Unix world with, essentially, 64-bit additions to the SUS, which would be implemented in every 64-bit version of Unix.

The careful reader will have already noted my use of the past tense. Welcome to reality in the Unix world. SCO and HP recently made it quite clear that they intend for their extensions and additions to be available only from SCO and HP. They will provide the whole Unix world with a unified 64-bit Unix, all right, *provided* that everyone uses *their* operating system on *their* hardware. Welcome back to market fragmentation—called “product differentiation” by the spin doctors.

Yes, This is Relevant

No, this isn't just a tale of woe. The Linux community used to be unified because of a simple lack of need for competition. Now, Linux distributions are somewhat differentiated, but most Linux vendors work together, recognizing that their long-term chances of survival are far better working together than fighting. Unlike the Unix community, the Linux community has stayed generally unified in the face of commercial interest.

Linux, though it started out as a toy, has for some time been real competition for “Real Unix”. Ignoring the licensing issues for the moment, Linux looks a lot like a vendor-differentiated version of Unix. While a few Unix versions have extra features that Linux does not (yet), such as journaling file systems, process migration, and fail-over server capability (See **Huh?**), Linux has features that distinguish it as well. For example, Linux has high-quality networking with support for many protocols; few commercial versions of Unix can provide Novell, Appletalk, SMB, and AX.25 in addition to standard TCP/IP networking.

Linux also uses memory frugally; with Linux, it is perfectly reasonable to use a machine with only 4MB of RAM—with most versions of Unix, that's not even enough to boot, let alone do useful work. Linux has more complete hardware support, especially for legacy hardware, than most (all?) versions of Unix for Intel x86 computers. Linux distributions usually include far more application software than is generally included in a Unix distribution. And, last but not least, Linux comes with source code.

Most versions of Unix support one, or at most two different CPU architectures. Sun's Solaris supports SPARC and Intel x86. SGI's Irix supports MIPS. SCO supports Intel x86. Digital Unix supports the Alpha. Linux currently supports Intel x86, Alpha, SPARC, Motorola 68K, PowerPC, MIPS, and Acorn ARM. The source code is now designed to make adding new architectures easy.

Linux is also now a 64-bit operating system. More properly, it is mostly bit-size-independent; it operates as a 32-bit operating system on a 32-bit CPU, a 64-bit operating system on a 64-bit CPU, and on a 16-bit CPU, the subset of Linux that can be fit into memory operates as a 16-bit operating system (see www.linux.org.uk/Linux8086.html).

Ignoring license issues, Linux and the various versions of Unix are pretty similar; they have the same core functionality, and each has a few extensions or features which differentiate it.

Potential

But licensing issues cannot be ignored. For every CPU that Linux supports, Unix or another clone is available. What Linux provides that Unix does not is a unified feature set across all the supported machines. The fact that Linux is licensed under the GPL means that a feature written on one architecture is available to the others, unless it is really by nature architecture-specific—and *very* few features are.

This feature-sharing among architectures helps give Linux the potential to outstrip Unix development in the long run. It certainly makes Unix vendors sweat, and with good reason. Linux has the long-term potential to cause Unix vendors to cooperate more fully with each other; if they refuse, Linux has the potential to eat away at a large part of their current markets.

Caveat venditor.

Archive Index Issue Table of Contents

Advanced search

Dissecting Interrupts and Browsing DMA

Alessandro Rubini

Georg V. Zezschwitz

Issue #26, June 1996

This is the fourth in a series of articles about writing character device drivers as loadable kernel modules. This month, we further investigate the field of interrupt handling. Though it is conceptually simple, practical limitations and constraints make this an “interesting” part of device friver writing, and several different facilities have been provided for different situations. We also investigate the complex topic of DMA.

Though last month's installment appeared to cover everything about interrupts, that is not true. One month later, you are ready to understand the ultimate technology for interrupt handling. We also begin to investigate the fascinating world of memory management by explaining the tasks of a DMA-capable driver.

Changes in Current Linux Versions

Before we start, you should note two changes in recent Linux versions. In Linux 1.3.70, the first steps were taken to support *shared interrupts*. The idea is that several devices and device drivers *share* the same interrupt line. This is also part of the PCI specification, where every device has its own vendor- and product-dependent device ID. When reading this ID, Linux gets quite verbose about plugged-in PCI devices when booting with PCI enabled. For this reason, the declaration of **request_irq** and **free_irq** in **linux/sched.h** now reads:

```
extern int request_irq(unsigned int irq,
   void (*handler)(int, void *, struct pt_regs *),
   unsigned long flags,
   const char *device,
   void *dev_id);
extern void free_irq(unsigned int irq, void *dev_id);
```

Thus, when registering an interrupt with **request_irq()**, a fourth parameter must be handed to Linux: the device ID. Currently, most devices will pass a **NULL** for

dev_id when requesting and freeing interrupts—this results in the same behaviour as before. If you really want to share interrupts, you also have to set **SA_SHIRQ** in **flags**, in addition to passing **dev_id**. Sharing of interrupts only works if all device drivers on one interrupt line agree to share their interrupt.

The second “change” is not a real change, but rather, a stylistic upgrade: **get_user_byte()** and **put_user_byte()** are considered obsolete and should not be used in new code. They are replaced by the more flexible **get_user** and **put_user** calls.

Linus Torvalds explains that these functions use compiler “magic” to look at the pointer they are passed and read or write the correct size item. This means you can't use **void *** or **unsigned long** as a pointer; you always have to use a pointer to the right type. Also, if you give it a **char ***, you get a **char** back, **not** an **unsigned char**, unlike the old **get_fs_byte()** function. If you need an **unsigned** value, use a pointer to an **unsigned** type. Never cast the return value to get the access size you want—if you think you need to do that, you are doing something wrong. Your argument pointer should always be of the right type.

Essentially, you should think of **get_user()** as a simple pointer dereference (kind of like ***(xxx)** in plain C, except it fetches the pointer data from user space instead). In fact, on some architectures, that is exactly what it is.

While we're fixing previous oversights, it is worth noting that the kernel offers a facility to autodetect interrupt lines. That's slightly different from what was shown in our article a couple of months ago. Those who are interested can look at **<linux/interrupt.h>** for documentation about it.

We now return you to your regularly scheduled programming.

A Split View of Interrupts

As you may remember from last time, interrupt handling is managed by a single driver function. Our implementation dealt with both low-level (acknowledging the interrupt) and high-level (such as awakening tasks) issues. This way of doing things may work for simple drivers, but it is prone to failure if the handling is too slow.

If you look at the code, it is clear that acknowledging the interrupt and retrieving or sending the relevant data is a minimal part of the workings. With common devices where you are moving only one or a few bytes per interrupt, most of the time is spent in managing device-specific queues, chains, and whatever other strange data structures are used in your implementation. Don't take our **skel_interrupt()** as an example here, since it is the most simplified interrupt handler possible; real devices may have a lot of status information and many modes of operation. If you spend too much time processing your data structures, it is possible to miss the next interrupt or interrupts and either hang or lose data, because when an interrupt handler is running, at least that interrupt is blocked, and with fast interrupt handlers, all interrupts are blocked.

The solution devised for this problem is to split the task of interrupt handling into two halves:

- First, a fast "top half" handles hardware issues and must terminate before a new interrupt is issued. Normally, very little is done here besides moving data between the device and some memory buffer (and not even that, if your device driver uses DMA), and making sure the hardware is in a sane state.
- Then a slower "bottom half" of the handler runs with interrupts enabled and can take as long as is needed to accomplish everything. It is executed as soon as possible after the interrupt is serviced.

Fortunately, the kernel offers a special way to schedule execution of the bottom half code, which isn't necessarily related to a particular process; this means both the request to run the function and the execution of the function itself are done outside of the context of any process. A special mechanism is needed, because the other kernel functions all operate in the context of a process—an orderly thread of execution, normally associated with a running instance of a user-level program—while interrupt handling is asynchronous and not related to a particular process.

Bottom Halves: When and How

From the programmer's point of view, the bottom half is very similar to the top half, in that it cannot call **schedule()** and can only perform atomic memory allocations. This is understandable, because the function is not called in the context of a process; the bottom half is asynchronous, just like the top half, the normal interrupt handler. The difference is mainly that interrupts are enabled and there is no critical code in progress. So, when exactly are those routines executed?

As you know, the processor is mostly executing on behalf of a process, both in user space and in kernel space (while executing system calls). The notable exceptions are servicing an interrupt and scheduling another process in place of the current one: during these operations the **current** pointer is not meaningful, even though it is always a valid pointer to a **struct task_struct**. Additionally, kernel code is controlling the CPU when a process enters/exits a system call. This happens quite often, as a single piece of code handles every system call.

With this in mind, it is apparent that if you want your bottom half executed as soon as possible, it must be called from the scheduler or when entering or leaving a system call, since doing it during interrupt service is taboo. Actually, Linux calls **do_bottom_half()**, defined in **kernel/softirq.c** from **schedule()** (**kernel/sched.c**) and from **ret_from_sys_call()**, defined in an architecture-specific assembly file, usually **entry.S**.

The bottom halves are not bound to the interrupt number, though the kernel keeps a static array of 32 such functions. Currently (I'm using 1.3.71), there is no way to ask for a free bottom half number or ID, so we will hard-code one. This is dirty coding, but it is used only to show the idea; later we will remove such ID stealing.

In order to execute your bottom half, you need to tell the kernel about it. This is accomplished through the **mark_bh()** function, which takes one argument, the ID of your bottom half.

This listing shows the code for a split-interrupt handler using the “dirty” ID pseudo-allocation.

```
#include <linux/interrupt.h>
#define SKEL_BH 16 /* dirty planning */
/*
 * This is the top half, argument to request_irq()
 */
static void skel_interrupt(int irq,
                           struct pt_regs *regs)
{
  do_top_half_stuff()
  /* tell the kernel to run the bh later */
  mark_bh(SKEL_BH);
}
/*
 * This is the bottom half
 */
static void do_skel_bh(void)
{
  do_bottom_half_stuff()
}
/*
 * But the bh must be initialized ...
 */
int init_module(void)
{
  /* ... */
  init_bh(SKEL_BH, do_skel_bh);
  /* ... */
}
/*
 * ... and cleaned up
 */
void cleanup_module(void)
{
  /* ... */
  disable_bh(SKEL_BH)
  /* ... */
}
```

Using Task Queues

Actually, dirty allocation of a bottom half ID is not really needed, because the kernel implements a more sophisticated mechanism which you will surely enjoy.

This mechanism is called “task queues”, because the functions to be called are kept in queues, constructed of linked lists. This also means you can register more than one bottom half function that is associated with your driver. Moreover, task queues are not directly related to interrupt handling and can be used independently of interrupt management.

A task queue is a chain of **struct tq_struct**'s, as declared in **<linux/tqueue.h>**.

```
struct tq_struct {
    /* linked list of active bh's */
    struct tq_struct *next;
    /* must be initialized to zero */
    int sync;
    /* function to call */
    void (*routine)(void *);
    /* argument to function */
    void *data;
};
typedef struct tq_struct * task_queue;
void queue_task(struct tq_struct *bh_pointer,
                task_queue *bh_list);
void run_task_queue(task_queue *list);
```

You'll notice the **routine** argument is a function getting a pointer argument. This is a useful feature, as you'll soon discover by yourself, but remember the **data** pointer is your complete responsibility: if it points to **kmalloc()**ed memory, you must remember to free it yourself.

Another thing to keep in mind is that the **next** field is used to keep the queue consistent, so you must be careful never to look in it, and *never* insert the same **tq_struct** in multiple queues, nor twice in the same queue.

There are a few other functions similar to **queue_task()** in the header, which are worth looking at. Here we stick to the most general call.

In order to use a task queue, you will need either to declare your own queue or add tasks to a predefined queue. We are going to explore both methods.

This listing shows how to run multiple bottom halves in your interrupt handler with your own queue.

```
#include <linux/interrupt.h>
#include <linux/tqueue.h#gt;
DECLARE_TASK_QUEUE(tq_skel);
#define SKEL_BH 16 /* dirty planning */
/*
 * Two different tasks
 */
static struct tq_struct task1;
static struct tq_struct task2;
/*
```

```
    * The bottom half only runs the queue
    */
   static void do_skel_bh(void)
   {
     run_task_queue(&tq_skel);
   }
   /*
    * The top half queues the different tasks based
    * on some conditions
    */
   static void skel_interrupt(int irq,
                              struct pt_regs *regs)
   {
     do_top_half_stuff()
     if (condition1()(I) {
       queue_task(&task1, &tq_skel);
       mark_bh(SKEL_BH);
     }
     if (condition2()(I) {
       queue_task(&task2, &tq_skel);
       mark_bh(SKEL_BH);
     }
   }
   /*
    * And init as usual
    */
   int init_module(void)
   {
     /* ... */
     task1.routine=proc1; task1.data=arg1;
     task2.routine=proc2; task2.data=arg2;
     init_bh(SKEL_BH, do_skel_bh);
     /* ... */
   }
   void cleanup_module(void)
   {
     /* ... */
     disable_bh(SKEL_BH)
     /* ... */
   }
```

Using task queues is an enjoyable experience and helps keep your code clean. For example, if you are running the skel-machine described in the previous few Kernel Korner columns, you can service multiple hardware devices by using a single interrupt handling function that gets the hardware structure as an argument. This behaviour can be accomplished by having a **tq_struct** as a member of the **Skel_Hw** structure. The big advantage here is if multiple devices request attention at nearly the same time, all of them are queued and serviced later all at once (with interrupts enabled). Of course, this only works if the Skel hardware isn't too picky about when interrupts are acknowledged and the interrupt condition is dealt with.

Running Predefined Queues

The kernel proper defines three task queues for your convenience and amusement. A custom driver should normally use one of those queues instead of declaring its own queue, and any driver may use the predefined queues without declaring new ones. The only reason there are special queues is higher performance: queues with smaller IDs are executed first.

The three predefined queues are:

```
struct tq_struct *tq_timer;
struct tq_struct *tq_scheduler;
struct tq_struct *tq_immediate;
```

The first is run in relation with kernel timers and won't be discussed here—it is left as an exercise for the reader. The next is run anytime scheduling takes place, and the last is run “immediately” upon exit from the interrupt handler, as a generic bottom half; this will generally be the queue you use in your driver to replace the older bottom half mechanism.

The “immediate” queue is used like **tq_skel** above. You don't need to choose an ID and declare it, although **mark_bh()** must still be called with the **IMMEDIATE_BH** argument as shown below. Correspondingly, the **tq_timer** queue uses **mark_bh(TIMER_BH)**, but the **tq_scheduler** queue does not need to be marked to run, and so **mark_bh()** is not called.

This listing shows an example of using the “immediate” queue:

```
#include <linux/interrupt.h>
#include <linux/tqueue.h>
/*
 * Two different tasks
 */
static struct tq_struct task1;
static struct tq_struct task2;
/*
 * The top half queues tasks, and no bottom
 * half is there
 */
static void skel_interrupt(int irq,
                           struct pt_regs *regs)
{
  do_top_half_stuff()
  if (condition1()(I) {
    queue_task(&task1,&tq_immediate);
    mark_bh(IMMEDIATE_BH);
    }
  if (condition2()(I) {
    queue_task(&task2,&tq_skel);
    mark_bh(IMMEDIATE_BH);
    }
}
/*
 * And init as usual, but nothing has to be
 * cleaned up
 */
int init_module(void)
{
  /* ... */
  task1.routine=proc1; task1.data=arg1;
  task2.routine=proc2; task2.data=arg2;
  /* ... */
}
```

An Example: Playing with tq_scheduler

Task queues are quite amusing to play with, but most of us lack “real” hardware needing deferred handling. Fortunately, the implementation of **run_task_queue()** is smart enough you can enjoy yourself even without suitable hardware.

The good news is **run_task_queue()** calls queued functions *after* removing them from the queue. Thus, you can re-insert a task in the queue from within the task itself.

This silly task prints a message only every ten seconds, up to the end of the world. It needs to be registered only once, and then it will arrange its own existence.

```
struct tq_struct silly;
void silly_task(void *unused)
{
  static unsigned long last;
  if (jiffies/HZ/10 != last) {
    last=jiffies/HZ/10;
    printk(KERN_INFO "I'm there\n");
  }
queue_task(&silly, &tq_scheduler);
/* tq_scheduler doesn't need mark_bh() */
}
```

If you're thinking about viruses, please hold on, and remember a wise administrator doesn't do anything as root without reading the code beforehand.

But let's leave task queues and begin looking at memory issues.

DMA on the PC—Dirty, Machine-dependent, Awful

Remember the old days of the IBM PC? Yes, those days, when the PC was delivered with 128 KB of RAM, an 8086-processor, tape interface and 360 KB floppy. [You got a whole 128KB? Lucky you! —ED] Those were the days when DMA on ISA bus was considered *fast*. The idea of DMA is to transfer a block of data from a device to memory or vice versa without letting the CPU do the boring job of moving individual bytes. Instead, after initialization of both the device and the on-board DMA controller of the motherboard, the device signals the DMA controller that it has data to transfer. The DMA controller puts the RAM in a state receiving data from the data bus, the device puts the data on the data bus, and when this has been done, the controller increments an address counter and decrements a length counter, so that further transfers go into the next location.

In those old days this technique was fast, allowing transfer rates of up to 800 kilobytes per second on 16-bit ISA. Today we have transfer rates of 132 MB/second with PCI 2.0. But good old ISA DMA has still its applications: imagine a sound card playing 16-bit coded music at 48 kHz stereo. This would result in 192 kilobytes per second. Transmitting the data by interrupt requesting, say, two words every 20 microseconds would certainly let the CPU drop a whole lot of interrupts. Polling the data (non-interrupt driven data transfer) at that rate would certainly stop the CPU from doing anything useful, too. What we need is a continuous data flow at midrange speed—perfect for DMA. Linux only has to initiate and stop the data transfer. The rest is done by the hardware itself.

We will deal only with ISA DMA in this article—most expansion cards are still ISA, and ISA DMA is fast enough for many applications. However, DMA on the ISA bus has severe restrictions:

- The hardware and the DMA controllers know nothing about virtual memory—all they can see is physical memory and its addresses (This restriction belongs to all kinds of ISA DMA.) Instead of using a page here and another there and gluing them together in virtual memory, we must allocate continuous blocks of physical memory, and we may *not* swap them out.
- Intel-based systems have two DMA controllers, one with the lower four DMA channels allowing byte-wise transfers and one with the upper four channels allowing (faster) word transfer. Both have only a 16-bit address counter and a 16-bit length counter. Therefore, we may not transfer more than 65535 bytes (or words, with the second controller) at once.
- The address counter represents only the lower 16 bits of the address (**A0**-**A15** on DMA channels 0-3, **A1**-**A16** on channels 4-7). The upper 8 bits of address space are represented in a *page* register. They will not change during a transfer, meaning transfers can take place only within one 16-bit (64 KB) segment of memory.
- Upper **8** bits? 24 bits of address space in total? Yes, it's sad, but true. We can only transfer within the *lower 16MB* of RAM. If your data eventually needs to reach another address, the CPU has to copy it “by hand” again, using the DMA-accessible memory as a “bounce buffer”, as it is called.

Allocating a DMA Buffer

Right, you know the restrictions—so decide now to read on or not!

The first thing we need for DMA is a *buffer*. The restrictions (lower 16 MB of memory, continuous page-addresses in physical memory) are all fulfilled if you allocate your buffer with:

```
void  *dma_buf;
dma_buf = kmalloc(buffer_size,
                  GFP_BUFFER | GFP_DMA);
```

The returned address will never fit the start of a page, as much as you would prefer it do so. The reason is Linux has a quite advanced memory management on used and unused blocks of pages with **kmalloc()**. It maintains lists of free segments as small as 32 bytes (64 on the DEC Alpha), another list for segments of double size, another for quadruple-size, etc. Every time you free **kmalloc()**ed memory, Linux will try to join your released segment with a free neighbor segment. If the neighbor is free, too, they are passed into the list of double size, where it is checked again, if it has a free-neighbor, to get into the next order list.

The sizes **kmalloc** supports at the moment (Linux 1.3.71) range from **PAGE_SIZE > 7** to **PAGE_SIZE << 5**. Each step in the power of 2 is one list, so two joined elements in the one list form one element of the next higher order.

You might wonder why you don't get a simple whole page. That's because at the beginning of every segment, some list information is maintained. This is called the (somewhat misleading) **page_descriptor**, and its size is currently between 16 and 32 bytes (depending on your machine type). Therefore, if you ask Linux for 64KB of RAM, Linux will have to use a block of free 128KB RAM and hand you 128 kilobytes to 32 bytes.

And this is difficult to get—the floppy driver sometimes dreams of this, when it tries to allocate its DMA buffer at run time and can't find any continuous RAM. Therefore, always think in powers of two, but then subtract some bytes (about 0x20). If you want to look at the magic numbers, they're all in **mm/kmalloc.c**.

The Role of Interrupts

Most devices using DMA will generate interrupts. For example, a sound card will generate an interrupt to tell the CPU, “Gimme data, I'm running out of it.”

The machine we built our driver for is quite fascinating: it is a laboratory interface with its own CPU, own RAM, analog and digital inputs and outputs, and bells and whistles. It uses a character channel for receiving commands and sending answers and uses DMA for block transfer of sampled data (or data to output). Therefore, an invoked interrupt might have these reasons:

- send data on the character channel
- read data from the character channel
- initiate a DMA-transfer to or from the host
- the transfer is done
- the transfer is going to cross the boundary of the DMA page register (DMA page-register has to be incremented)

Your interrupt handler has to find out what is meant by the interrupt. Normally, it will read a status register of the device having more detailed information on the task to perform.

As you see, we have gone far away from the general truth of a clean loading and unloading of a module and are right in the middle of dirty specialization. We decided our lab driver should perform the following tasks:

- Allocate DMA-able memory on user request and map this memory to the user space (the user has direct access to the DMA buffer and can “see” how it fills up—this is faster and more flexible than first capturing the data and then transferring it via, say, a character channel to the user).
- Check, whenever a transfer is required, if the buffer address is valid (was allocated by our driver) and if the length is sufficient.
- And, of course, do not forget to release the memory when the user closes the character channel, even if it has not been released explicitly.

This concept differs from the floppy driver, where you will never look directly at the actual DMA buffer. But it is probably good for you as well: you might decide to use this method for a frame grabber. The user might allocate multiple buffers, set up the transfer address for the one and look at the other while the first is filled up. As the only thing the user and the system have to do is toggle the sampling address and the buffers are both mapped into the user address space, not a single byte of the picture has to be transferred by the CPU—they just arrive at the location where the user wants them. We will describe the tricks to do this in the next Kernel Korner.

Before we start with real code, let us think of the steps to be taken for a complete transfer:

1. An interrupt occurs, meaning a transfer should start.
2. The interrupt handler starts the transfer.
3. The interrupt handler returns, while the CPU starts its normal activity and the transfer is running.
4. Interrupt occurs, meaning the transfer is finished.
5. The interrupt handler ends the transfer...
6. ...and probably asks the device for another one, wakes up *sleeping beauties*, etc.

Don't be disappointed that we don't write your whole device driver—things will be very different in different situations! Here's the code for step 2) and 5):

```
static int skel_dma_start (unsigned long dma_addr,
                           int dma_chan,
                           unsigned long dma_len,
                           int want_to_read) {
    unsigned long flags;
    if (!dma_len || dma_len > 0xffff)
        /* Invalid length */
        return -EINVAL;
    if (dma_addr & 0xffff !=
       (dma_addr + dma_len) & 0xffff)
```

```
            /* We would cross a 64kb-segment */
            return -EINVAL;
        if (dma_addr + dma_len > MAX_DMA_ADDRESS)
            /* Only lower 16 MB */
            return -EINVAL;
        /* Don't need any irqs here... */
        save_flags (flags); cli ();
        /* Get a well defined state */
        disable_dma (dma_chan);
        clear_dma_ff (dma_chan);
        set_dma_mode (dma_chan,
                      want_to_read ?
                      /* we want to get data */
                      DMA_MODE_READ
                      /* we want to send data */
                     : DMA_MODE_WRITE);
        set_dma_addr (dma_chan, dma_addr);
        set_dma_count (dma_chan, dma_len);
        enable_dma (dma_chan);
        restore_flags (flags);
        return 0;
    }
    static void skel_dma_stop (int dma_chan) {
        disable_dma (dma_chan);
    }
```

Sorry, we can't give you a more detailed code here, as you have to decide for yourself how to include DMA in your driver. As usual, the best way to get things working is to look at some working implementation as a starting point.

Deeper and Further

If you want to go deeper with the topics just described, the best teacher is the source, as usual. Split-half interrupt handlers and task queues are used throughout the mainstream kernel, while the DMA implementation shown here is taken from our **ceddrv-0.17**, available by ftp from **tsx-11.mit.edu**.

The next installment will come back to more concrete issues—we realize that both DMA and task queues may appear to be rather esoteric topics. We'll show how **mmap()** works and how a driver may implement its semantics.

Alessandro Rubini is a 27-year-old Linuxer with a taste for the practical side of computer science and a tendency to delay sleeping. This helps him meet deadlines by exploiting time-zone offsets.

Georg V. Zezschwitz is also 27-year-old Linuxer with the same taste for the practical side of computer science and a tendency to delay sleeping.

Linux in the Real World

Joe Klemmer

Issue #26, June 1996

Linux in the U.S. Army, or How I Spent My Winter Vacation

The United States Army Publications and Printing Command (USAPPC) is, as implied by its name, that part of the Army charged with the creation, publication, and distribution of all Army publications. This can include everything from simple forms up to complete technical manuals. The Command uses CD-ROMs to distribute a list of some 40,000 publications to the field units, who use them as a basis for submitting orders for publications. These CDs are mailed out quarterly, but the process of getting the CDs cut and distributed can take as long as four or five months. See the problem? By the time some customers get their copy of the CD, it has already been replaced by the next one, and can potentially contain obsolete listings. How could this dilemma be solved? Linux, of course!

An ideal platform for distributing or ordering publications is the World Wide Web. The Command already has a TCP/IP link to the Internet; all it would take is a system to run as the server. The Command had looked into putting up a web server in the summer of 1995 but decided against it because they were told it would cost $60,000 to implement. This is where I became involved. After being informed of the situation in November 1995, I asked if I could attempt to set up a web server on one of the Pentiums the Command had. After being told that, due to budget constraints, no money could be spent on this project, I was given a PC. Now for the fun stuff.

Like most Linux enthusiasts, I have a number of CDs with various versions of Linux distributions at home. I brought them in and set to work. First, I installed the Fall release of Slackware 3.0. After figuring out the type of Ethernet card installed (nothing like blind guessing), the installation went smooth as silk. But there were some minor problems running 3.0, so I dropped back to version 2.3 for my base. I installed and configured the entire system in half a day. (It was easy after doing it hundreds of times on my home system.)

Now to find an HTTP server. I looked at the usual choices: NCSA, CERN, Apache... All are good programs, but I ended up going with WN. WN is a fast, flexible HTTPD that has built in search and image map capabilities as well as very strong security. It can be found at ftp://ftp.acns.nwu.edu/pub/wn/. Further information may be found at WN's home page: hopf.math.nwu.edu/. The main reasons for this choice were its easy installation and the built-in search engine, which would be perfect for what was needed. Once I had the system up and working, I started building the pages.

It turned out that constructing the web site was more effort than loading the OS. My first task was to set up an ordering system for publications. Since the application to process orders was on the mainframe, I set up a form that takes the necessary input and saves it to a file. Then, every night, a cron job sends the contents of this file to the mainframe using NCFTP. This way, the current system—with all its editing and security checks—can be used, and the procedures for submitting orders by e-mail and paper that were already in place did not have to change.

Next came the task of putting the contents of the publications CD-ROM on the web site. Using the program that came with the CD, I generated extract files for all the different types of publications. This totaled 7 files of over 39 MB. I put these on the server, and using the built-in search capabilities of WN, created a form to view and/or search the files for user-defined strings.

Once this was working, I started work on having a job run on the mainframe that would extract the publications data in the correct format from the original source that was used to make the CDs. This can be retrieved via NCFTP as often as needed so the current data is always available on the web site.

Right now, I'm working with the section that produces forms in electronic format. These forms, which are in Perform Pro and Formflow formats, are also distributed to customers on CD-ROM. I am currently building a page where customers can search and download the forms they need using FTP. This should be working by the time this article is published.

Future plans for this system include linking it up with a dial-up BBS so that customers without direct Net access will be able to access the ordering and search systems with the data shared between the BBS and the web site. From there, who knows? If you'd like to see what has been done on the USAPPC site, the address is www-usappc.hoffman.army.mil.

Because of this system, the cost savings for publications ordering and distribution will be quite large. All this was made possible by Linux; without Linux, there would be no USAPPC web site at all.

And I'd still be hacking away at JCL.

Joe Klemmer (klemmerj@webtrek.com) is a 33-year-old civilian Informations Systems employee of the US Army, and has worked for them for over 10 years. A follower of Linux since version 0.12, he enjoys giving away Linux CDs to spread the faith. Other than Linux, his passions include his wife, Joy, and their four ferrets and six finches (as of this writing).

Caldera Network Desktop 1.0

Phil Hughes

Issue #26, June 1996

This article offers a look at the capabilities of CND and its market niche.

The Caldera Network Desktop (CND) is not just another Linux distribution. Some people will be very happy to hear this; others will be disappointed. Rather than an in-depth product review, this article offers some first impressions and talks about where CND is a good fit.

Starting in Fall 1993, with the first Yggdrasil release, Linux has been evolving in the commercial arena. That is, prior to that time, most newcomers found Linux on the Internet or in the hands of a friend and worked from there. Yggdrasil changed that, and other vendors (including Walnut Creek, CraftWorks, and Red Hat) continue to offer commercial products.

Like Yggdrasil, a pioneer that took Linux into a different arena, Caldera has done it again, expanding the possible Linux market into a whole new area. What needs to be addressed is: what is this new market, and who, along with Caldera, will need to get involved to service this market? But before we get into that, here are my initial experiences with installation.

Installation

When I opened the package I found a manual, a CD in a jewel case, a very skinny mouse pad (that gets mixed reviews in our office), and two floppy disks. I was pleased to find the floppies but very surprised there were only two, since Red Hat (which Caldera is based on) uses two root disks plus a boot floppy that you must select from 72 possible choices.

It turns out Caldera is using a new installation system and only one floppy is required. One of the two is for standard computers, the other is for laptops with PCMCIA cards.

I quickly thumbed through the manual as I searched for some free disk space on biggie, the machine I have used to load and test most of the Linux distributions on the planet. The documentation looks good, covering most of what you need to know without becoming a text for techno-nerds.

I slipped in the boot disk and CD and booted up the system. It came up to a boot screen where I entered my one required boot-time parameter so my SCSI controller could be found. All worked well and I was ready to start answering questions.

Without going into great detail, LISA (Linux Installation and System Administration) is a text-mode installation system that was very easy to use and asked the right number of questions. By that I mean, if something needs to be asked, it is, but users aren't pelted with questions about things they might not care about or understand.

All went well until the X installation. At that point the system refused to believe I had an S3-based video board and kept defaulting to VGA mode. The board I have is a brand-X, but I have it running fine under Slackware, Red Hat and Yggdrasil.

I finally gave up and changed to a name-brand video board that was one of the choices on the X configuration menu and all worked fine. The X server is X Inside, which is, I understand from talking to others, very good and fast. While it supports cards that are not supported by the free server (like the Matrox line) it has a reputation for not liking some clone cards.

At this point, I decided I needed to try installing CND on another computer to see how well it worked. I chose my Toshiba T3600 notebook which is currently running Slackware 2.3. I plugged in a New Media BusToaster card and SCSI CD-ROM drive and booted it up from the PCMCIA floppy. After booting, it asked me to insert my PCMCIA card, wait five seconds and press return. I did. It successfully identified the card, but then died because the module to talk to the card was not on the boot disk. While I could have found the module, made another boot disk, and tried again, I decided to quit and hope that version 1.1 addresses that problem.

Once It Is Running…

On boot after installation, all came up fine. My only complaint is that it starts about every possible server and daemon in the world. While this does make it easy for people who don't know what they want, it also means editing rc files for those who need to streamline their system.

Once CND is up and running, it offers a drag and drop desktop. But, you already know that from the reviews of the preview version and the Caldera ads. So, once again, I wanted to make it do all the stuff I had been doing with other Linux distributions, and my first effort was to get my PPP link back up and running.

There was no GUI-based setup for PPP, so I looked in the manual. Sure enough, PPP was in the index. The reference basically said to read the PPP HOWTO (which is on the CD). This wasn't fatal, as I merely copied the PPP stuff I had under Slackware over and it worked fine. But again, there was nothing easier about setting it up under CND.

The New Market

While the information above may sound depressing, I don't really feel that it is. While Joe Average may want to convince his wife that CND is a good buy because it looks like MS Windows, I don't think this is the primary market. If Caldera hired me to be their marketing manager, I would tell them their markets are the office desktop and custom office situations. They don't have to hire me to do this, though, as it appears they already know. For example, they have established Value Added Reseller (VAR) and Independent Software Vendor (ISV) programs.

If I wanted to get Linux into the office market, I could pick a particular segment of the office market (like education, medical office, etc.), develop the packages needed (probably some sort of database), get up to speed installing CND, and start selling a bunch of software.

Better yet, pick some specific hardware and get into the market with a total hardware/software solution. Also, since CND includes commercial NetWare client and administration tools, getting CND on desktops in a Novell-based office makes a lot of sense.

In conclusion, CND is not another Slackware or Red Hat. It is designed for the end-user, “I don't know what a Linux is” market. With one hundred million or more PCs out there, CND has a very good chance to move Linux into new places.

Archive Index Issue Table of Contents

Advanced search

Acucobol

Robert Broughton

Issue #26, June 1996

I used Acucobol in an SCO Xenis environment in a previous job. I was happy with the product, so all I had to do to evaluate the Linux version was make sure it works as well as the Xenix one.

Yes, I've read the flame wars that have appeared in just about every comp.* newsgroup about COBOL. They are irrelevant to this article. In evaluating Acucobol as a product, I'm interested in whether the statements:

```
move 2 to x.
add 3 to x.
```

produce a value of five, not whether it's better or worse to code:

```
x = 2;
x += 3:
```

instead.

I used Acucobol in an SCO Xenix environment in a previous job. I was happy with the product, so all I had to do to evaluate the Linux version was make sure that it works as well as the Xenix one.

Like most commercial products currently available for Linux, Acucobol has been ported to a large number of platforms. Acucobol goes a step further. The Acucobol compiler actually translates the COBOL source into a “b-code” file. When you execute an Acucobol program, you actually run a b-code interpreter. It's no problem for a program compiled on one sort of machine to be executed on any other machine that runs Acucobol. You can compile programs on Linux and run them on Sun workstations, Vaxen, or even MS-DOS or Windows NT. (Yes, Acucobol is smart enough to convert **/** characters in paths to ****. I'm not sure what happens when a component of the path is longer than eight characters, however.)

If your last exposure to COBOL was in a mainframe, batch processing environment, you will see some major differences. Acucobol is not very fussy about which column statements begin in. There are some significant new language features. One that I particularly like is the **EVALUATE** statement, which is similar to the *switch* statement in C. There are also a lot more places where an **ELSE** statement can be used. It's possible to reference substrings using what is known in COBOL as “reference modification”; the form is *data-name* (*leftmost-position*: *length*). The **SCREEN SECTION**, the windowing capability, and the ability for an **ACCEPT** statement to bring in an entire screen of data are Acucobol extensions. It's even possible to reference “embedded procedures” within the screen section.

Another new feature is a subprogram named **W$MENU**. Older versions of Acucobol came with a subprogram named **MENUBAR**. It was written entirely in COBOL and worked very nicely. **W$MENU** has all of **MENUBAR**'s capabilities, except one. **W$MENU** is designed so that your menus are always activated by a hot key. Suppose you want the menu to appear whenever your program requires a menu choice The only way I could figure out to do it was to call another built-in subprogram, **W$KEYBUF**, which stuffs the input buffer. I used **W$KEYBUF** to stuff the hot key into the input buffer, and the menu appeared. (Acucobol Inc. has promised to provide a cleaner way of doing this in the future.) **W$MENU** is complemented by a rudimentary menu generator; it's batch, not interactive.

It's possible to develop C functions that can be called by Acucobol, and the manual does a good job of explaining how to do this. The only problem is, the functions must be linked into the b-code interpreter.

Acucobol uses neither termcap nor terminfo. Instead, it uses a file named a_termcap, which you would normally install in the /etc directory. The supplied a_termcap file contains an xterm entry, but this entry had no color or graphics support. I was able to create a color xterm entry by cannibalizing other a_termcap entries. It ended up looking like this:

```
xterm-c|color xterm:\
        :GO=\E(0:GF=\E(B:GM=qxlkmjvtwun:VB:\
        :C1=\E[30m:C2=\E[34m:C3=\E[32m:\
        :C4=\E[36m:C5=\E[31m:C6=\E[35m:\
        :C7=\E[33m:C8=\E[37m:\
        :B1=\E[40m:B2=\E[44m:B3=\E[42m:\
        :B4=\E[46m:B5=\E[41m:B6=\E[45m:\
        :B7=\E[43m:B8=\E[47m:UL@:RU@:tc=xterm:
```

Acucobol comes with a reference manual and a user's guide. They are both loose-leaf and about 1,000 pages apiece. The manuals are used for all Acucobol implementations, including Unix, MS-DOS, and VMS.

Like any other COBOL implementation, Acucobol supports sequential, indexed, and “relative” (direct-access) files. Acucobol's indexed file system is called Vision, which is proprietary to Acucobol. Vision files have keys and data combined into a single file, and the index structure is b-tree. Vision is efficient and reliable. The only problem is that if you want non-Acucobol programs to access Vision files, you will have to buy a license from Acucobol Inc. A more interesting solution is an Acucobol add-on called “Plug and Play”. It works like this: All COBOL ISAM statements (**READ NEXT**, **START**, etc.) are mapped into a dispatch table. In the standard Acucobol run-time, the table entries point to Vision functions. You can replace this table with a different one, so that the table entries point to D-ISAM or Codebase functions. I have an evaluation copy of Codebase (the subject of my next review) on my desk, so I plan to actually try this.

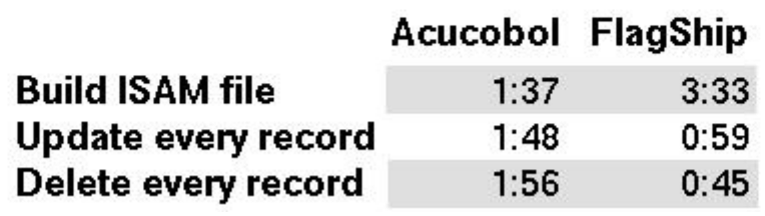
Acucobol should be capable of compiling and running any standard-compliant COBOL program. However, unless the program has no user interface whatsoever, you're probably going to want to do some conversion work. In particular, Acucobol has an RM/COBOL compatibility mode. It works perfectly well, but my experience has been that it's still better to rearrange the way in which data is displayed on and accepted from terminal screens, because Acucobol allows you to do this in a less tedious fashion.

There's a substantial pile of application software and add-ons available for Acucobol. Two worth mentioning are the MCBA and Real World accounting packages.

A Benchmark

I developed a simple benchmark for Acucobol, which is a translation of another benchmark I did for xBASE products. (See *Linux Journal* #14, June 1995, p. 27.) This benchmark processed a file containing 33,830 records and was run on a 66 mHz 486 with a SCSI disk.

It's reasonable to expect that Acucobol would be noticeably slower than Flagship, since Flagship programs are compiled into machine code, and Acucobol programs are compiled into b-codes, which must be interpreted at run time.



	Acucobol	FlagShip
Build ISAM file	1:37	3:33
Update every record	1:48	0:59
Delete every record	1:56	0:45

Figure 1. Benchmark Results

Other Observations

Acucobol Inc. hosts a developers' conference in San Diego every year. The next one is September 18-20. I haven't been to one of these yet, but I've had good reports from them; attendees are treated well by Acucobol, Inc.

There are references in the manuals to GUI and mouse support, but they seem to be only for Windows NT. Acucobol, Inc. does seem to be interested in offering some sort of UNIX GUI support eventually.

This product will probably not be of much interest to hard-core Linux people; they would rather program in C for free than pay big bucks to use COBOL. It should be of interest to the Linux community, though, that Acucobol, Inc. went to the trouble of porting their product, because it gives Linux more legitimacy.

However, the Linux version of Acucobol should be of great importance to another community: system houses that develop and maintain COBOL application software. When these system houses sell a product to an end user, the end user must not only purchase the application, s/he must also purchase the hardware the application runs on and the rest of the software environment. For example, if the application runs under Windows 95 or SCO Unix, either the customer or the systems house must buy a copy of Windows 95 or SCO Unix. A system house that has a COBOL application to sell can now try to convince customers to accept Linux as the software environment instead and can offer a lower total cost to the customer.

Robert Broughton (roberb7@iceonline.com) has been developing software for 24 years and has been using Linux since February, 1993. He is employed by Zadall Systems Group, in Burnaby, BC, Canada. His web page is at: www.iceonline.com/home/roberb7/WWW/index.html.

New Products

Phil Hughes

Issue #26, June 1996

Synchronize 2.0 Available For Linux, Accelerated X 1.3 Now Available and more.

Synchronize 2.0 Available For Linux

CrossWind Technologies has introduced its Synchronize 2.0 real-time enterprise collaboration software for Linux. Synchronize's support for Linux enables enterprise network managers to deliver enterprise-scale scheduling and task management in mixed computing environments. Designed for cross-platform deployment in a client/server environment communicating across TCP/IP networks, Synchronize is available for Windows NT Server, commercial Unix servers, and Linux, as well as all enterprise desktops, including MS Windows, Windows NT, Macintosh, X11/Motif, and ASCII. Price: $100.00 per user.

Contact: CrossWind Technologies, Inc., 1505 Ocean Street, Suite 1, Santa Cruz, CA 95060. Phone: 1-408-469-1780. Fax: 1-408-469-1750. E-mail: info@crosswind.com. URL: www.crosswind.com.

Accelerated X 1.3 Now Available

X Inside has announced shipment of a new version of its high performance Accelerated X for Unix. Accelerated X is a display server product from X Inside that supports nine different Unix operating systems, including Linux. Accelerated X 1.3's enhancements over the previous version includes supporting over 400 graphics cards compared to 347 in the previous version.

Contact: X Inside, Inc., 1801 Broadway, 17th floor, Denver, CO 80202. Phone: 1-800-X-INSIDE or 1-303-298-7478. Fax: 1-303-298-1406. E-mail: sales@xinside.com. URL: www.xinside.com.

Dynace Object Oriented Extension to C Available

Algorithms Corporation has announced the full source code release of the Dynace Object Oriented Extension to C. There is no charge for the source code for personal, educational and evaluational purposes, and it may be downloaded from Algorithms Corporation's Internet site at www.edge.net/algorithms. Dynace is a preprocessor, include files and a library which extends the C or C++ languages with advanced object oriented capabilities, automatic garbage collection and multiple threads. Dynace comes with full C source and is portable to 16- and 32-bit DOS, Windows 3.1, Windows 95, Windows NT, Linux, SunOS, and others. Dynace can link with pre-existing C/C++ libraries and can be easily added to pre-existing C/C++ code.

Contact: Algorithms Corporation, 3020 Liberty Hills Drive, Franklin, TN 37067. Phone: 1-800-566-8991 or 1-615-791-1636. Fax: 1-615-791-7736. E-mail: Dynace-info@edge.net. URL: www.edge.net/algorithms.

EditTable and ChartObject Available

Interactive Network Technologies, Inc. (INT) has announced Linux versions of its popular table and charting tools. INT's EditTable Widget and ChartObject Library provide Linux programmers with flexible, reliable tools for creating, displaying and editing tables and charts. EditTable contains resources for interactive control of all aspects of table data visualization and manipulation. ChartObject includes a comprehensive library of easy-to-use 2D and 3D graphing tools for building presentation-quality charts and graphs. Freeware Linux versions of both EditTable and ChartObject are available from INT's Web site at www.int.com/linux.html. Commercial versions of these products are also available for both Unix and Linux platforms.

Contact: Interactive Network Technologies, Inc., 2901 Wilcrest, Suite 300, Houston, TX 77042-6011. Phone: 1-713-975-7434. Fax: 1-713-975-1120. E-mail: info@int.comi. URL: www.int.com/linux.html.

VBIX Available For Linux

Halcyon Software has announced the release of VBIX, a Microsoft Visual Basic 3.0 compatible runtime engine for Unix environments, including Linux. VBIX allows Visual Basic applications developed in the Windows 3.1 environment to be executed directly on Unix platforms running Motif. Price: $495.00 per user.

Contact: Halcyon Software, 1590 La Pradera Dr., Campbell, CA 95008. Phone: 1-408-378-9898. Fax: 1-408-378-9935. E-mail: dhsi@vbix.com, sales@vbix.com. URL: www.vbix.com.

Running Linux Companion CD-ROM

O'Reilly & Associates and Red Hat Software, Inc., have announced the release of the Running Linux Companion CD-ROM. The two CD set contains Red Hat 2.1 and supporting documentation. When paired with O'Reilly's *Running Linux* book, the CD/book combination provides a complete software/documentation package for installing and learning to use the Linux operating system. Price: $24.95 for Companion CD-ROM.

Contact: O'Reilly & Associates, phone: 1-800-998-9938 or 1-800-889-8969. E-mail: nuts@ora.com. URL: www.ora.com.

Archive Index Issue Table of Contents

Advanced search

Consultants Directory

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Ian T. Zimmerman
Linux consulting.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Archive Index Issue Table of Contents

Advanced search