Linux Journal Issue #90/October 2001

Features

Open-Souce Software at the Aerodynamics Laboratory *by Steve Jenkins*
Jenkins uses a variety of open-source software to keep wind tunnel data flowing smoothly.
Linux and Samba in a Federal Lab *by Brian Gollsneider and Mike Martin*
Users at this army research lab think they're accessing data from an NT fileserver—but it's Linux.

Toolbox

Take Command at Your Service—Job Scheduling for Linux *by Louis J. Iacona*
Kernel Korner How to Write a Linux USB Device Driver *by Greg Kroah-Hartman*
At the Forge Data Modeling with Alzabo *by Reuven M. Lerner*
Cooking with Linux Engineering Intelligence *by Marcel Gagné*
Paranoid Penguin GPG: the Best Free Crypto You Aren't Using, Part II of II *by Mick Bauer*
GFX Alias|Wavefront Maya 4 *by Robin Rowe*

Columns

Linux in Education Modeling Seismic Wave Propogation on a 156GB PC Cluster *by Dimitri Komatitsch and Jeroen Tromp*
Focus on Software Distribution Upgrades *by David A. Bandel*

Geek Law Naming Open-Source Software *by Lawrence Rosen*
Linux for Suits The Bazaar Way to Bet *by Doc Searls*
Focus on Embedded Systems The Robots Are Coming, The Robots Are Coming *by Rick Lehrbaum*

Reviews

Microlite BackupEDGE Version 01.01.08 *by Charles Curley*

Departments

Letters
upFRONT
From the Editor *by Richard Vernon*
Best of Technical Support
New Products

Strictly On-Line

O'Reilly Show Report, Day One *by Doc Searls*

Archive Index

Advanced search

Open-Source Software at the Aerodynamics Laboratory

Steve Jenkins

Issue #90, October 2001

Steve describes a typical aircraft experiment and the open-source software involved.

Long ago, before the Open-Source Software (OSS) movement, before the World Wide Web, before the Free Software Foundation and GNU, I was hired by the Unsteady Aerodynamics Laboratory of the National Research Council of Canada to work with the data-acquisition system for their high-speed wind tunnel. At that time, the lab had a specialized real-time minicomputer: a Hewlett-Packard HP-1000 F series. Once a year, through the early- and mid-1980s, I would go to the HP International Users Group conferences and return home with a magnetic tape containing the contributions of the attendees. Mounting that tape and looking through the index file, I felt like a kid unwrapping presents on Christmas morning. This was my first exposure to source code sharing, and I had no idea what the future would hold for such a simple concept.

Canadian National Ski Team Member in the 2m × 3m Tunnel

In 1990 I acquired my first UNIX box, a then state-of-the-art Silicon Graphics 4D/80GT, along with my own T1 connection to the Internet. The switch from a small real-time OS on a standalone computer to an IRIX-based networked machine opened the door to a brave new world with a very steep learning curve. By 1992 I was writing Byzantine scripts that used combinations of shell, awk and sed to manipulate wind tunnel data files. One day when I was on Usenet looking for advice, someone mentioned Perl. I now wish that I had taken notice of whoever it was so that I could thank them for making my life so much easier. Within a year Perl had become indispensable on the SGI as well as on my desktop Macintosh. It would become the most important piece of open-source software at the laboratory.

About that time I experimented with a program that, although it seemed somewhat useful, I underestimated rather badly at first. It was called Mosaic. A short while later, I installed the NCSA's HTTPd and began to understand the potential of the Web. In its later incarnation as Apache, it would become the second largest OSS project upon which the lab depends daily.

The Institute for Aerospace Research was restructured in 1995, and after the dust had settled I found myself at my current location: the 2 × 3 meter, low-speed wind tunnel of the Aerodynamics Laboratory. At that time, I began creating web-based software to extend the capabilities of the existing data system that had relied on command-line and X Window System user interfaces running under QNX and AIX. The decision to switch to browser-based programs was due in large part to the type of clients coming to the tunnel. Even though the bulk of our work is aircraft testing for companies such as Bombardier Aerospace, over the past several years we have tested cars, buses, trucks, motorcycles (my favorite), power lines, bridges, antennae, as well as Olympic cyclists, skiers and bobsled teams. This wide variety of clients, with computer skills that range from inept to adept, makes user-interface design challenging. Since even management knows how to surf the Web, I decided to try a web-based interface written in Perl on one of our applications. The feedback I received was overwhelmingly positive due to the ease-of-use and high comfort level experienced by our clients and staff, so I continued to build web-based tools.

Vehicle Aerodynamics in the 2m x 3m Tunnel

The third significant OSS project to be adopted came several years later. In 1998 we purchased a 24-node Alpha/Linux Beowulf cluster for our computational fluid dynamics group. This was a good project for evaluating new technology because, although this task is much more computationally intensive than the wind tunnel data system, it is not as critical on a daily basis to the lab's clients. The success we experienced with this installation convinced us that Linux was a viable alternative to the commercial operating systems we had been using.

While these big pieces were falling into place, we also began using several smaller OSS applications on a regular basis: Ghostscript, Xmgr, Vim and NEdit, to name just a few.

The Present

In order to provide a context for the rest of this article, I'll describe a typical aircraft experiment from the data-processing point of view. After a model is installed in the wind tunnel it usually takes from one to five weeks to complete a test. During that time up to 500,000 independent measurements will be taken. This can result in the creation of as many as 2,000 X-Y plots, 4,000 disk files and as much as 500MB of text data to be displayed on-screen. It is imperative to have fast and simple methods of dealing with all this information.

The clients and test engineers of the laboratory have access to all of the facility's data-acquisition, storage and visualization systems through Perl CGI programs running under Apache (see Figure 1). Much of the control of the experiment by the wind tunnel operators is also handled the same way. When a user opens the web browser on any of the computers in the control room, the wind tunnel client home page is automatically loaded. This page allows access to the five web-based software tools I have written so far: plotting, configuration file editing, data file viewing, event logging and the dynamic data display (see Resources). In addition, there are also links to local resources such as system documentation and a unit-conversion calculator, as well as to off-site information. I'd like to point out that the laboratory uses a rather restrictive intranet model that helps to alleviate some of the security concerns of running web-based systems.

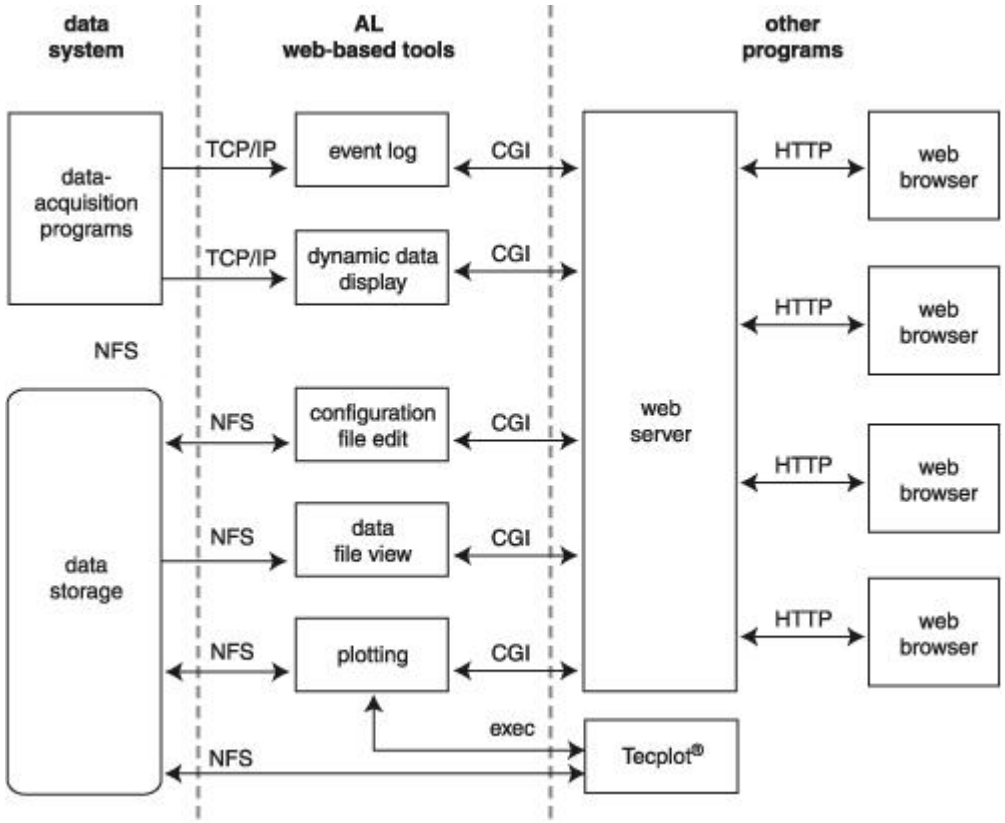


Figure 1. Block Diagram of the Web-Based Software

The plotting system was the first of the software tools to be developed, and as such, was used as a proof-of-concept for the idea of providing clients of the wind tunnel with access to their data through the Web. Since we were already using a commercial data visualization package from Amtek Engineering called Tecplot, I decided to build the plotting system around it. My software allows users to set up plotting templates simply by selecting options and filling in text boxes in an HTML form. These templates are used to generate Tecplot command files that can be utilized immediately to produce plots either on-screen in the browser or on paper. A dæmon program (also written in Perl) also uses those same templates to produce paper plots automatically, at the end of each wind tunnel run.

Spinning F18 Model in the Water Tunnel

Configuration file editing is accomplished through another web-based program. It was created to provide a fast and simple method of modifying the files that control the programs for acquiring and reducing the experimental data. Users are shown a form where each line contains a parameter name with either a text box or a select element to set its value. The Perl program that generates these HTML forms also dynamically generates JavaScript code to validate the user input before it is submitted. If any invalid entries are detected, flashing arrows appear next to the input fields and a popup dialog box describes the specific problems.

The data file viewer is a simple CGI program that searches through the disk space for a given wind tunnel test. It creates an HTML button for each entry found that matches our naming convention for data and configuration files. These buttons are presented to the user in tables where each row corresponds to one tunnel run and each column to a specific data type (e.g., raw, reduced, tare, etc.). Each press of a button brings up a new browser window with the contents of the selected file parsed and formatted for viewing. Then, users have the option of downloading the file to their local computer in CSV, Matlab or one of several other formats.

All of the new software and much of the legacy code generates status messages—events that are handled by an event-logging system that consists of two main parts. The first is a simple Perl dæmon that listens on a TCP/IP port for messages and stores them in log files. The second part of the system is a web-based viewer that allows users to search the log files for events that match specific criteria such as time of occurrence, computer name, event severity level, etc. Although this seems like a trivial application, it is indispensable because the data-acquisition, manipulation and visualization system consists of several computers running heterogeneous operating systems. Finding bugs in this kind of distributed system (especially timing problems due to complex interactions) is difficult if not impossible without a common event log.

Generic Fighter Aircraft Model in the 2m × 3m Tunnel

From the user's perspective, the dynamic data display system is the only noninteractive software tool. It is based on a Perl server that accepts data

messages from the data-acquisition systems. Users can view these messages by connecting to the server through a CGI program that uses nonparsed headers (NPH) or “server push”. This program presents the user with a data table and dynamically adds new information to the top of a table as it becomes available. The old data is scrolled down and eventually goes off the end of the table. While creating this code I was somewhat concerned about the possibilities of memory leaks, not only in Perl or Apache, but also in our browsers. I shouldn't have been. We have had individual NPH clients that have remained continuously connected to the server throughout wind tunnel tests that have lasted more than five weeks. During that time they displayed over 500MB of data with no problems.

Individually, each of these five tools works well enough but is hardly revolutionary. When they are combined, however, they form a simple, consistent and robust environment for our clients and staff to interact with their experiment. There is no need to remember obscure commands, long data paths, command key sequences or any of the other things associated with many types of user interfaces. All users need to do is point and click and fill in the blanks on web pages, something they know how to do and with which they are comfortable.

The Future

The first thing on my to-do list is to finish moving the Perl code I've developed off our last remaining SGI and onto a dual-processor Intel/Linux system. Although getting the programs running in their current form is a trivial task, I'm using the opportunity to refactor all of the code. There is also one software tool that is partially developed and needs to be finished: the user interface to the model attitude control system. In addition, we are considering changing our data file format from an arcane system developed in-house to one using XML. This will certainly result in the need for more new code.

Peering further into the future, I hope to have the time to develop some VRML applications that create dynamic 3-D graphic simulations of the models and probes in our wind tunnels, with superimposed load and pressure data. Also, our instrumentation group is investigating using embedded/real-time Linux for some of our data-acquisition needs.

With all of this work still to be done, I have no doubt that open-source software will continue to play an increasingly important role in the day-to-day operation of the Aerodynamics Laboratory.

Resources

Steve Jenkins is the senior programmer/analyst at the Aerodynamics Laboratory of the Institute for Aerospace Research, National Research Council of Canada and has over 20 years experience with data processing in aerospace facilities.

Archive Index Issue Table of Contents

Advanced search

Linux and Samba in a Federal Lab

Brian Gollsneider

Mike Martin

Issue #90, October 2001

Using Linux and Samba for research on extremely small lasers called VCSELs.

Linux and Samba recently answered the needs of the Army Research Lab (ARL) at Adelphi, Maryland. Our branch does state-of-the-art research into a specific type of lasers and amasses large amounts of data during the performance testing of these devices. We were able to connect our test equipment over the network to a Samba server. The twist to this approach is that our configuration makes it appear to the users that they access the data through the branch's NT fileserver. I'll explain the setup in detail, but the main trick is creating a network shortcut on the NT box to point to the Samba share while making the Linux box invisible on the network. Figure 1 depicts the setup of the network.

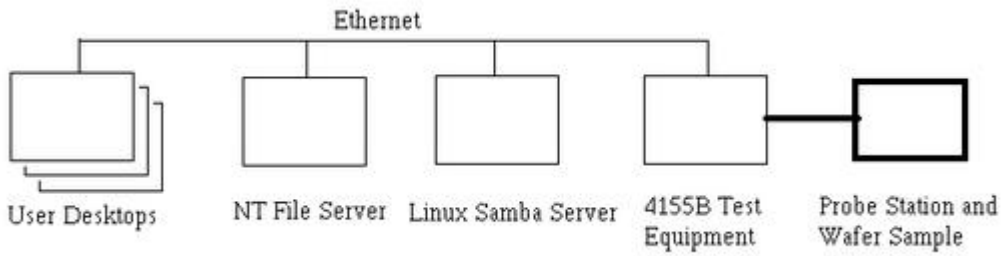


Figure 1. Network Setup

Our branch develops extremely small lasers called VCSELs (vertical-cavity, surface-emitting lasers), which fall under the general category of photonics research. We easily can put over 60 lasers into a square millimeter, and the full wafer containing the lasers can be three inches in diameter. Therefore, we can have thousands of devices on a single wafer. Figure 2 shows a picture of a typical VCSEL. The main tests we run to characterize the performance of each VCSEL are called ILV curves for current, light and voltage. Basically, we see how much light comes out for the power that was put in. Also, most of the analysis software is on the user's desktop machine so they need to be able to access the

raw data from there. Users are creatures of habit. Getting data pertinent to the branch has historically meant going to the NT server. Since the users were used to getting data from the NT box, we did not want to make them go somewhere else. We tried to make everything transparent to the user and make it appear as though they were getting the data from the NT server. To force the users to go through the NT box, we make the Linux box invisible to the network. We rely on the security of the NT box to authenticate users accessing the data.

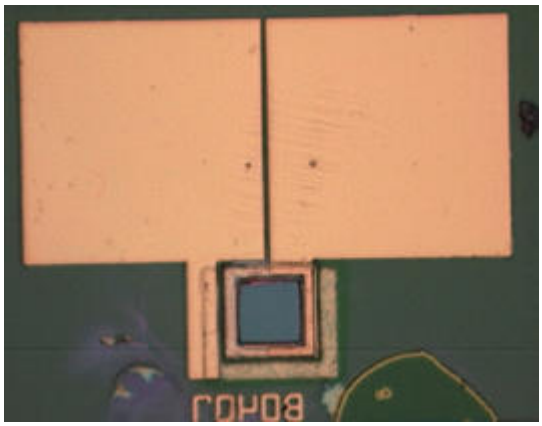


Figure 2. Typical VCSEL: large rectangles are contact pads for the test probes. The actual lasing area is the small gray square at bottom center.

Test Setup

Two pieces of equipment are key to characterizing the VCSELs. First is the probe station that is basically just a microscope with some tiny probes and a light meter. The probes apply the power to the device, and we measure the power produced with the light meter. A 4155B parameter analyzer from Agilent is the second piece of equipment. This analyzer is programmed to sweep the current level and measure the voltage and light. It has two main ways of being controlled: front panel and the GPIB interface. Granted, the GPIB port is a popular scientific interface and allows us to do fancier tests by controlling the test setup with a computer as well as collect the data, but our controlling computer is about five feet down the lab bench and cannot be moved closer. This makes it difficult to start the test when the probes are in place. Fortunately our main test is simple to program through the front panel. Our test routine is to position the probes by looking through the eyepiece of the microscope, reach up carefully and push the test button on the parameter analyzer and then save the data. Figure 3 shows the lab hardware.

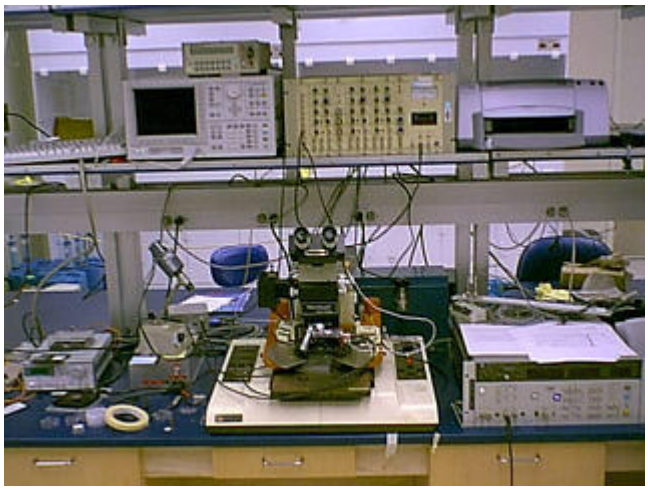


Figure 3. Probe Station (bottom center) and 4155B Parameter Analyzer (White Box on top left)

Operation

After we get a clean run, we need to save the data. The 4155B has three ways to save the data: GPIB, floppy and TCP/IP. Since we aren't controlling the analyzer with the GPIB, that's not an option. The floppy supports 3.5" disks, but these disks fill up quickly and you have to walk around with them. Since we have several lab areas where we work, it's not unheard of to have to backtrack to recover a temporarily misplaced disk. The answer we put together works because of the TCP/IP support.

Linux

The parameter analyzer supports TCP/IP, specifically NFS. You can even ping the analyzer. Since it's registered in the lab's DNS, the ping can be done by way of IP address or name. We were able to put together a Linux box out of obsolete or broken equipment. Literally, we pulled together parts of three computers into one. It didn't cost the government anything, and it fills the need. For the installation, the newest distribution that we had and that the P-133 hardware would support is Red Hat 6.2, so we put that on and hardened it with Bastille and the latest patches. Additionally, all the unnecessary services were turned off and SSH was added. We sliced the hard drive space carefully and ended up with about 1.5GB of space for data. Total time of install and configuration was three hours.

NFS

Again, the parameter analyzer talks NFS, so the next step was configuring that. The /etc/exports file needed just one line:

```
/home/guest/hptestdata 192.168.10.29(rw)
```

The hptestdata directory was created under guest's home directory, and nfsd was restarted. This line allows only the one IP address to mount the directory.

Appropriate information was entered into the parameter analyzer's front panel and the mount button pushed. Of course it didn't work the first time. After just a minute of diagnosis, syncing the ID numbers on the analyzer to the guest account solved the problem. Total time to configure NFS was less than five minutes.

Samba

Samba is an amazing product that can do many things. This is a simple application, and the /etc/smb.conf is shown in Listing 1.

Listing 1. /etc/smb.conf

Of course, crucial security information like network domain has been changed in this and /etc/exports. The key parts in the file are creating the hptestdata share and making it read-only. The read-only part is to prevent users from accidentally deleting data. We periodically purge, but only after assurances from all the users. The other part of Samba is modifying the boot-up files so nmbd is killed. With the network configuration we are setting up, we don't want to see the machine on the network. Therefore we don't want nmbd to provide name services. See your distribution's documentation for the appropriate file to configure. For Red Hat 6.2, we modified S91smb and commented out the nmbd startup lines by placing a # at the beginning of the appropriate lines. To remind myself of this network configuration, I also changed the echo line in the file to say that smbd was not starting. Normally the script will output that nmbd is starting. Access is restricted to our domain only so outside access is prevented. Total time to configure was several hours of tweaking.

NT Setup

The final configuration step was on the NT box. We haven't seen this trick anywhere else so we think it's pretty neat. We created a data share for the Linux machine. This is where the users will go for data from their desktops. Then we made a network shortcut using UNC (universal naming convention) and put it into the data share. To be honest, to do this we made the Samba share visible on the network for just a minute and created a shortcut in the directory. It was easier for us to do that than fight getting the double backslashes correct. When the user accesses the NT server, he or she sees the shared folder. Double-clicking there shows a directory. Double-clicking on the directory brings the user to the Linux box with the test data, without realizing it. This trick is necessary because Windows cannot share out a network drive that it has mounted. My original plan was to have the NT box map the Samba share to a drive and then share that out. Total time to configure was five minutes, after realizing that Windows can't share out a mapped drive and we employed this trick.

Conclusion

Linux and Samba filled a requirement of the lab that couldn't be supported otherwise. The method is transparent to the users because they go to the same central place for data; it's as secure as the branch's NT server, and it was literally built for free since we used 100% scavenged equipment.

Future

This scheme still suffers slightly from security. A savvy computer user could look at the properties of the network shortcut and then use that to make a shortcut directly to the Samba server, bypassing the NT security. An alternative would be to use the Linux box and smbmount to mount a share from the NT server and export that using NFS to the test device. We were able to mount the NT share on the Linux box, export that with NFS and then mount that on the 4155B. The problem still remaining is writing to that share, even using options with smbmount. Hopefully, in the near future we will have some time to tackle this aspect again.

Resources

Brian Gollsneider is a student at the University of Maryland working on a PhD in electrical engineering. He works with the researchers at the Army Research Laboratory investigating VCSELs. Brian Gollsneider may be reached via e-mail at gollsneb@glue.umd.edu.

Mike Martin is a student at the University of Maryland working on a BS in electrical engineering. He works with the researchers at the Army Research Laboratory investigating VCSELs.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

At Your Service—Job Scheduling for Linux

Louis J. Iacona

Issue #90, October 2001

A simple command-line utility for planning and managing deferred program executions.

Drawing from the communications world, services fall in one of two broad categories, immediate or deferred. Immediate communication services include teleconferencing, on-line chat, etc.; while deferred services include e-mail and fax. This column presents an overview of the Linux **at** utility that allows for the deferred (or scheduled) execution of a program.

Of course, most programs need to be launched in an immediate, do-it-now mode. Still, it's often desirable and perhaps required to schedule the execution of a program (or job, in ancient Geek) at a particular time for a variety of practical reasons. Consider any of the following scenarios:

1. A development group has developed a script that rebuilds their source tree and updates the application's staging area. The script is completely mechanized, but it is very time- and CPU-intensive. As a practical matter, the group decides this job should be scheduled to run in the late evening when it would be competing with fewer users and processes.
2. A high-quality color printer/plotter is in high demand during regular business hours. Multiple times a week, a project manager uses the printer to reprint a quickly evolving project plan of 200+ pages. To avoid monopolizing the printer during its peak usage period, the project manager decides the print job should be run in the early morning—before most users are active.
3. A project leader wants to distribute meeting reminder/agenda notes via e-mail to participating staff members four hours before a meeting scheduled for next Friday at 4:00 **P.M**.
4. A system administrator (SA) has been receiving reports of extremely poor system performance at the start of the business day. The cause of the

degrading performance is not obvious. Rather than comb through numerous system logs, the SA decides to schedule the execution of a program that captures system load information. The job is to be executed the next day, every five minutes during the 9:00 **A.M.** hour.

5. An external data source used to update a local data warehouse becomes available at 6:00 **P.M.** every Friday. If the lengthy update procedure does not complete successfully, it's to be retried an hour later. Otherwise, it is rescheduled for the next Friday.

The scheduling requirements outlined in each of the above scenarios can be addressed by the at utility. Furthermore, these scenarios represent a broad range of the most common applications for at. The at utility is ideal for performing CPU-intensive tasks when little user-processing demand is present on the system, utilizing scarce resources at a time when they're generally more available, distributing reminders at a specific time, executing jobs that need to run at a time when the user has no intention of being actively connected to the system and executing jobs that are dependent on resources becoming available at a specific time.

at's Features

at is actually a collection of related programs that allow Linux users to schedule and manage the execution of deferred jobs. If you're thinking at is only suitable for noninteractive tasks, you are correct. Programs scheduled through at are expected to be capable of running as background processes, since at does not associate user-display devices with executing jobs. Users generally find the at command-line interface to be quite intuitive. Yet, it is expressive enough to meet any conceivable scheduling requirement. Before presenting examples of the command-line interface, let's summarize the basic services provided by at. It allows for scheduling the execution of a job at a specific time/date, displaying information about jobs currently scheduled, canceling a job currently scheduled and administrating the list of users that have permission to use at programs.

Using at

The examples presented here assume you have a basic familiarity with shell command execution and Bourne shell syntax. Each of the examples were tested on a Red Hat Linux 7 (2.3 kernel) platform. However, I would expect the examples to work as presented on all common distributions.

Obviously, the most fundamental capability offered by at is that of scheduling the execution of a program at some later time. Generally installed at /usr/bin/at, the basic command-line syntax for the at program is as follows:

```
at [options] TIME [DATE] < bourne-shell-file
```

This syntax implies the following alternate ways of invoking at would work as well:

```
cat bourne-shell-file | at [options] TIME [DATE]
at [options] TIME [DATE]
## Anything normally accepted by the bourne shell
## interpreter will be accepted. Terminate with
## Ctrl-D (^d)
statement 1
statement 2
statement 3
statement 4
^d
```

The first significant detail here is the proper notation for the time and date. The date is always optional. When it's omitted from the command line, the date is assumed to be the next day the specified time will occur, that is, either today or tomorrow. For example, if the time is specified as 1:30 **P.M.** and it's already 4:30 **P.M.**, the job will be scheduled for the next day at 1:30 **P.M**. **at** accepts a time/date notation that extends the POSIX.2 standards. That notation may be documented on your installation under the /usr/doc directory in a file called timespec.

Let's learn through some examples. The time/date specification examples in Table 1 will give you a good idea of what the at program will accept.

Table 1. Possible Time/Date Entries

As shown, the at program accepts a fairly rich and intuitive notation. It attempts to interpret the time/date specified by parsing its command-line arguments from left to right. If its time/date specification is violated, a “Garbled time” error diagnostic is displayed< and the program terminates. It usually provides a terse hint as to why it became troubled. For example, review the following invocation attempt below:

```
$ at 6am Mar 32
Error in day of month. Last token seen: 32
Garbled time
$
```

The following invocation succeeds at scheduling a job. Consider the third usage scenario outlined above for this submission:

```
$ at 1pm friday
warning: commands will be executed using /bin/sh
    STAFF="moe larry curly"
    cd mail
    mail -s"Meeting Reminder" $STAFF < friday_agenda.txt
    ^d
job 6 at Fri Apr  13 13:00:00 2001
$
```

If you're following closely, some questions should begin to formulate. Do the standard output and standard error streams of a scheduled job get captured somewhere? All standard output and standard error diagnostics are captured by the at service (/usr/sbin/atd program) and e-mailed to the submitting user when the scheduled job finishes. The e-mail will appear with a subject heading such as the following:

Subject: Output from your job 17

Are we limited to scheduling Bourne shell scripts? As the feedback from the example above indicates, at uses the system Bourne shell program (/bin/sh) to interpret the user-provided program statements. Therefore, anything you would type at a Bourne shell prompt would be valid, including a Bourne shell script or the launch of any executable found in your environment, even a different language interpreter. Review the following example for scheduling the execution of a Perl script:

```
$ at 6pm tomorrow
warning: commands will be executed using /bin/sh
    perl /home/moe/perls/script1.pl
    ^d
job 28 at Wed Apr  18 18:00:00 2001
$
```

What system users/groups get assigned to the scheduled process? What attributes of the job submission environment are preserved and carried over into the job execution environment? Discovering how a utility provides a service behind the curtains is usually not of interest to most users. In the case of at, however, an actual example of what the utility does is fairly simple to follow and shows what's possible, what isn't and why. Every scheduled job results in a generated Bourne shell script placed under the /var/spool/at/ directory. These generated scripts are comprised of the following sections:

- Program comments that provide some clues as to which user is notified when the job completes. Also, the effective user and group ID assigned to the job is specified.
- umask setting dictates how new files/directories should be created.
- A full listing of environment variable assignments at the time the job was submitted (except those related to display devices, such as TERM and DISPLAY).
- The current directory changes to where the user was when the task was scheduled. If that directory does not exist when the job executes, the job aborts and an e-mail notification is sent.
- An appended copy of the program text that was submitted to at.

With this in mind, Listing 1 is an abbreviated example of a script generated on behalf of a job scheduled by our user Moe. To complete our command-line interface description of the at program, see Table 2 for some of its more useful command-line options.

Listing 1. Sample—Moe's Job

Table 2. Command-Line Options

The following is another example of scheduling a job with at:

```
$ at -mf /home/curly/shells/program1  6pm tomorrow
warning: commands will be executed using /bin/sh
job 29 at Thu Apr  12 18:00:00 2001
$
```

Two other programs are tied to the basic at program: atq and atrm. As you might guess, atq lists jobs currently scheduled while atrm cancels one or more specified jobs. While the root user is capable of viewing and canceling any and all jobs, nonroot users can only view and cancel what they schedule.

Listing 2 shows an example of the output displayed by atq, as viewed by the root user (I inserted the field labels for better readability—unfortunately the Linux implementation of atq does not provide them).

Listing 2. Sample atq Output

Rank is a unique sequence value used to identify scheduled jobs to other at programs. In addition to the scheduled time and the submitting user, a queue value is listed. Unless the user specified otherwise, jobs are placed on the “a” queue. (Reference the on-line manual page for the implications of specifying an alternate queue value to the at program—it essentially controls the runtime priority of the job.) **atq** qualifies currently running jobs with a “=” value in the queue field. So job 17, seen in Table 3, is currently executing.

What if a user submits a job and then realizes either the time and/or program is incorrect? The atrm utility can be used to remove one or more scheduled jobs. For example, the root user could cancel the first and second jobs listed in Table 3 with the following command:

```
atrm 18 19
```

atrm provides no feedback. A subsequent atq listing would then display a queue like the one in Listing 3.

Listing 3. Revised Sample atq Output

Administering at

If your attempt to use one of the at programs produces the following diagnostic: “You do not have permission to use at”, you need to contact your local SA. Linux SAs can manage at with a fair amount of flexibility.

As you might expect, the root user has absolute permission to use the at utility and can grant the same permission to nonroot users. Two system files, /etc/at.allow and /etc/at.deny, control access to the at utility. Table 3 shows how their presence and content determines users' permission on a given system.

Table 3. Controlling User Access to at

Note that if the /etc/at.allow files exist, /etc/at.deny is completely ignored. Users are identified in both files by their Linux login, each appearing on a separate line. **at** does not provide a command-line utility to control the content of these files. SAs generally select their favorite text editor and manually edit the files as needed. This hardly can be considered a shortcoming, though, given the likely infrequency of change.

To summarize, nonroot users can be explicitly or implicitly assigned or denied permission to use at. SAs either can choose to manage access to at by exclusion or inclusion. Select the approach that makes the most sense for your particular installation. For example, a highly sensitive production site probably should be managed based on inclusion (i.e., nonroot users do not have permission unless it's explicitly granted—the /etc/at.allow file exists). Conversely, the Linux default configuration might be fine for most development/test environments (i.e., nonroot users have permission unless it's explicitly denied—/etc/at.allow does not exist and /etc/at.deny has zero or more entries).

Conclusion

Collectively, the at programs offer an intuitive way to manage the deferred execution of applications. Despite its simplicity and usefulness, the at utility is often ignored by Linux administrators and developers. Other less frequently used at command-line options exist that I chose not to cover here. I encourage you to review the at manual page by typing **man at** at your favorite shell's prompt to review all details. Also, most Linux overview books provide some coverage of at and similar programs, such as O'Reilly's *Linux in a Nutshell* by Ellen Siever, et. al.

Miscellaneous at Facts

Louis J. Iacona (lji@omnie.com) has been designing and developing applications on Linux/UNIX since 1982. He is currently a senior staff member of OmniE Labs, Inc. (www.omnie.com).

Archive Index Issue Table of Contents

Advanced search

How to Write a Linux USB Device Driver

Greg Kroah-Hartman

Issue #90, October 2001

Greg shares his USB driver skeleton and shows how it can be customized for your specific device.

The Linux USB subsystem has grown from supporting only two different types of devices in the 2.2.7 kernel (mice and keyboards), to over 20 different types of devices in the 2.4 kernel. Linux currently supports almost all USB class devices (standard types of devices like keyboards, mice, modems, printers and speakers) and an ever-growing number of vendor-specific devices (such as USB to serial converters, digital cameras, Ethernet devices and MP3 players). For a full list of the different USB devices currently supported, see Resources.

The remaining kinds of USB devices that do not have support on Linux are almost all vendor-specific devices. Each vendor decides to implement a custom protocol to talk to their device, so a custom driver usually needs to be created. Some vendors are open with their USB protocols and help with the creation of Linux drivers, while others do not publish them, and developers are forced to reverse-engineer. See Resources for some links to handy reverse-engineering tools.

Because each different protocol causes a new driver to be created, I have written a generic USB driver skeleton, modeled after the pci-skeleton.c file in the kernel source tree upon which many PCI network drivers have been based. This USB skeleton can be found at drivers/usb/usb-skeleton.c in the kernel source tree. In this article I will walk through the basics of the skeleton driver, explaining the different pieces and what needs to be done to customize it to your specific device.

If you are going to write a Linux USB driver, please become familiar with the USB protocol specification. It can be found, along with many other useful documents, at the USB home page (see Resources). An excellent introduction to the Linux USB subsystem can be found at the USB Working Devices List (see

Resources). It explains how the Linux USB subsystem is structured and introduces the reader to the concept of USB urbs, which are essential to USB drivers.

The first thing a Linux USB driver needs to do is register itself with the Linux USB subsystem, giving it some information about which devices the driver supports and which functions to call when a device supported by the driver is inserted or removed from the system. All of this information is passed to the USB subsystem in the usb_driver structure. The skeleton driver declares a usb_driver as:

```
static struct usb_driver skel_driver = {
     name:        "skeleton",
     probe:       skel_probe,
     disconnect:  skel_disconnect,
     fops:        &skel_fops,
     minor:       USB_SKEL_MINOR_BASE,
     id_table:    skel_table,
};
```

The variable name is a string that describes the driver. It is used in informational messages printed to the system log. The probe and disconnect function pointers are called when a device that matches the information provided in the id_table variable is either seen or removed.

The fops and minor variables are optional. Most USB drivers hook into another kernel subsystem, such as the SCSI, network or TTY subsystem. These types of drivers register themselves with the other kernel subsystem, and any user-space interactions are provided through that interface. But for drivers that do not have a matching kernel subsystem, such as MP3 players or scanners, a method of interacting with user space is needed. The USB subsystem provides a way to register a minor device number and a set of file_operations function pointers that enable this user-space interaction. The skeleton driver needs this kind of interface, so it provides a minor starting number and a pointer to its file_operations functions.

The USB driver is then registered with a call to usb_register, usually in the driver's init function, as shown in Listing 1.

Listing 1. Registering the USB Driver

When the driver is unloaded from the system, it needs to unregister itself with the USB subsystem. This is done with the usb_unregister function:

```
static void __exit usb_skel_exit(void)
{
   /* deregister this driver with the USB subsystem */
   usb_deregister(&skel_driver);
}
module_exit(usb_skel_exit);
```

To enable the linux-hotplug system to load the driver automatically when the device is plugged in, you need to create a MODULE_DEVICE_TABLE. The following code tells the hotplug scripts that this module supports a single device with a specific vendor and product ID:

```
/* table of devices that work with this driver */
static struct usb_device_id skel_table [] = {
    { USB_DEVICE(USB_SKEL_VENDOR_ID,
      USB_SKEL_PRODUCT_ID) },
    { }                     /* Terminating entry */
};
MODULE_DEVICE_TABLE (usb, skel_table);
```

There are other macros that can be used in describing a usb_device_id for drivers that support a whole class of USB drivers. See usb.h for more information on this.

When a device is plugged into the USB bus that matches the device ID pattern that your driver registered with the USB core, the probe function is called. The usb_device structure, interface number and the interface ID are passed to the function:

```
static void * skel_probe(struct usb_device *dev,
unsigned int ifnum, const struct usb_device_id *id)
```

The driver now needs to verify that this device is actually one that it can accept. If not, or if any error occurs during initialization, a NULL value is returned from the probe function. Otherwise a pointer to a private data structure containing the driver's state for this device is returned. That pointer is stored in the usb_device structure, and all callbacks to the driver pass that pointer.

In the skeleton driver, we determine what end points are marked as bulk-in and bulk-out. We create buffers to hold the data that will be sent and received from the device, and a USB urb to write data to the device is initialized. Also, we register the device with the devfs subsystem, allowing users of devfs to access our device. That registration looks like the following:

```
/* initialize the devfs node for this device
   and register it */
sprintf(name, "skel%d", skel->minor);
skel->devfs = devfs_register
              (usb_devfs_handle, name,
               DEVFS_FL_DEFAULT, USB_MAJOR,
               USB_SKEL_MINOR_BASE + skel->minor,
               S_IFCHR | S_IRUSR | S_IWUSR |
               S_IRGRP | S_IWGRP | S_IROTH,
               &skel_fops, NULL);
```

If the devfs_register function fails, we do not care, as the devfs subsystem will report this to the user.

Conversely, when the device is removed from the USB bus, the disconnect function is called with the device pointer. The driver needs to clean any private

data that has been allocated at this time and to shut down any pending urbs that are in the USB system. The driver also unregisters itself from the devfs subsystem with the call:

```
/* remove our devfs node */
devfs_unregister(skel->devfs);
```

Now that the device is plugged into the system and the driver is bound to the device, any of the functions in the file_operations structure that were passed to the USB subsystem will be called from a user program trying to talk to the device. The first function called will be open, as the program tries to open the device for I/O. Within the skeleton driver's open function we increment the driver's usage count if it is a module with a call to MODULE_INC_USE_COUNT. With this macro call, if the driver is compiled as a module, the driver cannot be unloaded until a corresponding MODULE_DEC_USE_COUNT macro is called. We also increment our private usage count and save off a pointer to our internal structure in the file structure. This is done so that future calls to file operations will enable the driver to determine which device the user is addressing. All of this is done with the following code:

```
/* increment our usage count for the module */
MOD_INC_USE_COUNT;
++skel->open_count;
/* save our object in the file's private structure */
file->private_data = skel;
```

After the open function is called, the read and write functions are called to receive and send data to the device. In the skel_write function, we receive a pointer to some data that the user wants to send to the device and the size of the data. The function determines how much data it can send to the device based on the size of the write urb it has created (this size depends on the size of the bulk out end point that the device has). Then it copies the data from user space to kernel space, points the urb to the data and submits the urb to the USB subsystem (see Listing 2).

Listing 2. The skel_write Function

When the write urb is filled up with the proper information using the FILL_BULK_URB function, we point the urb's completion callback to call our own skel_write_bulk_callback function. This function is called when the urb is finished by the USB subsystem. The callback function is called in interrupt context, so caution must be taken not to do very much processing at that time. Our implementation of skel_write_bulk_callback merely reports if the urb was completed successfully or not and then returns.

The read function works a bit differently from the write function in that we do not use an urb to transfer data from the device to the driver. Instead we call the

usb_bulk_msg function, which can be used to send or receive data from a device without having to create urbs and handle urb completion callback functions. We call the usb_bulk_msg function, giving it a buffer into which to place any data received from the device and a timeout value. If the timeout period expires without receiving any data from the device, the function will fail and return an error message (see Listing 3).

Listing 3. The usb_bulk_msg Function

The usb_bulk_msg function can be very useful for doing single reads or writes to a device; however, if you need to read or write constantly to a device, it is recommended to set up your own urbs and submit them to the USB subsystem.

When the user program releases the file handle that it has been using to talk to the device, the release function in the driver is called. In this function we decrement the module usage count with a call to MOD_DEC_USE_COUNT (to match our previous call to MOD_INC_USE_COUNT). We also determine if there are any other programs that are currently talking to the device (a device may be opened by more than one program at one time). If this is the last user of the device, then we shut down any possible pending writes that might be currently occurring. This is all done with:

```
/* decrement our usage count for the device */
--skel->open_count;
if (skel->open_count <= 0) {
   /* shutdown any bulk writes that might be
      going on */
   usb_unlink_urb (skel->write_urb);
   skel->open_count = 0;
}
/* decrement our usage count for the module */
MOD_DEC_USE_COUNT;
```

One of the more difficult problems that USB drivers must be able to handle smoothly is the fact that the USB device may be removed from the system at any point in time, even if a program is currently talking to it. It needs to be able to shut down any current reads and writes and notify the user-space programs that the device is no longer there (see Listing 4).

Listing 4. The skel_disconnect Function

If a program currently has an open handle to the device, we only null the usb_device structure in our local structure, as it has now gone away. For every read, write, release and other functions that expect a device to be present, the driver first checks to see if this usb_device structure is still present. If not, it releases that the device has disappeared, and a -ENODEV error is returned to the user-space program. When the release function is eventually called, it determines if there is no usb_device structure and if not, it does the cleanup

that the skel_disconnect function normally does if there are no open files on the device (see Listing 5).

Listing 5. Cleanup

This usb-skeleton driver does not have any examples of interrupt or isochronous data being sent to or from the device. Interrupt data is sent almost exactly as bulk data is, with a few minor exceptions. Isochronous data works differently with continuous streams of data being sent to or from the device. The audio and video camera drivers are very good examples of drivers that handle isochronous data and will be useful if you also need to do this.

Writing Linux USB device drivers is not a difficult task as the usb-skeleton driver shows. This driver, combined with the other current USB drivers, should provide enough examples to help a beginning author create a working driver in a minimal amount of time. The linux-usb-devel mailing list archives also contain a lot of helpful information.

Resources

Greg Kroah-Hartman is one of the Linux kernel USB developers. His free software is being used by more people than any closed-source projects he has ever been paid to develop.

Archive Index Issue Table of Contents

Advanced search

Data Modeling with Alzabo

Reuven M. Lerner

Issue #90, October 2001

Reuven takes a detour this month and shows you how to bridge the object-relational gap.

Over the last few months, we have been looking at server-side Java programming from a variety of perspectives. From servlets to JSPs to the Enhydra application server, we've seen several different ways to create dynamic, database-driven web sites using open-source Java technologies.

I had originally planned to continue in that vein this month, looking at Enhydra's intriguing DODS object-to-relational modeling software. DODS provides a high-level Java abstraction layer for tables in a relational database. DODS methods are translated automatically into the appropriate SQL, which is then handed to the database. The result: you see Java objects and methods, your database sees tables and SQL, and everyone is happy.

Unfortunately, the help and goodwill expressed by folks at Lutris (the corporate backer of Enhydra) were no match for the Israeli customs service and our local branch of FedEx. The CD and book with additional explanations of DODS sit in a warehouse as I write this, forcing me to take a short detour from my original plan.

However, investigating DODS for this month's article revived my interest in the subject of object-relational mappings. One of the most interesting and easy-to-use tools that I've seen for this purpose is Alzabo, a set of Perl modules that allows server-side Perl programmers to wrap their relational database schemata inside of an object. (The project is named for a creature in the science fiction work of Gene Wolfe.) I was quite impressed by what I saw and believe that many Perl programmers will be equally happy to discover such a powerful tool.

The Problem

Programmers have reaped many benefits by working with objects, from reusability to inheritance to encapsulation. But while programmers have taken to object-oriented programming in droves, object databases have been less popular for a variety of reasons. Instead, relational databases have become increasingly popular over the last few years, with huge quantities of data being placed within them. The problem, then, is how we can model our data as objects, while storing them as tables.

One possibility is to model each table as a class, each table column as an instance variable and each table row as an instance of that class. But anyone who has tried this quickly discovers it is easier said than done, particularly when creating web applications—how can we join two tables? What happens when two programs modify the same row in memory and only later commit those changes to the database? How can we ensure that changes to our class definition are reflected in the database and vice versa?

Another possibility is to read an entire table into an object instance, modifying the object and writing it out when a particular method is invoked. This works pretty well for small tables, but what happens when your tables become several megabytes (or gigabytes or terabytes) in size? Your boss might be willing to buy more memory for the web server but not if you're wasting it all reading entire tables into memory! Besides, modeling tables inside of your object means you also have to create a decent locking mechanism, complete with commits and rollbacks—something that most programmers are equipped to do.

We can easily dismiss these problems when working on a small application. But as applications and databases scale up, we want to ensure that things will work as expected. This is particularly true when creating an object-to-relational mapping system, such as Alzabo. One of my employees and I created a simple object-to-relational mapping middleware layer last year and were very happy with what we had done—until we found that we hadn't taken nearly enough corner cases into account, ending up with a mess of exceptions and default values.

Luckily for the Perl programmers among us, Dave Rolsky took the time to sit down and map out all of these problems, as well as many others. Alzabo gives us an object-oriented middleware layer that removes our need to interact directly with a database.

But Alzabo does more than provide a high-level interface to your database. It also gives you a programmatic way to modify your database schema definitions, including a browser-based table creation and maintenance tool that

creates SQL for you automatically. Moreover, Alzabo can take an existing database and reverse-engineer it, allowing you to use Alzabo with existing databases as well as new ones.

Installing Alzabo

Like most Perl modules, Alzabo is available for download from CPAN. However, installing Alzabo can be more complicated than other modules, simply because Alzabo depends on many modules. Not only does Alzabo require the use of DBI (for database access) and either DBD::mysql or DBD::Pg (for PostgreSQL), but the browser-based schema-creation tool uses HTML::Mason, which in turn requires mod_perl. If all of these are installed on your system, then installing Alzabo should be relatively straightforward.

I was able to install Alzabo without too much difficulty, using the CPAN modules to download and install automatically each of the prerequisites and then Alzabo itself.

I accepted the default values for almost all of the questions asked during the software's configuration and installation, with the exception of the .mhtml suffix that Alzabo assumes you use for Mason components. I normally give Mason components the simple .html suffix; because my Apache configuration didn't know what to do with the .mhtml extension, it sent them as Content-type text/plain, displaying the Mason component's source code in my browser window. Changing the suffix of the installed Mason components to .html worked on my computer, but I could have modified my Mason or Apache configuration just as easily.

Alzabo tracks each schema in its own directory, called /usr/local/alzabo by default. Inside of this directory is a schemata directory, with a single subdirectory for each of the database schemata that Alzabo is modeling. For example, the appointments schemata would be in /usr/local/alzabo/schemas/appointments.

There were two small hitches in my Alzabo installation that I had to fix. First, I had to change the permissions /usr/local/alzabo so that my web user could read and write to it. Secondly, I had to modify my PostgreSQL startup script to include the -i option, so that clients could connect via the network. By default, most PostgreSQL installations (including RPM versions) do not turn on -i, meaning that even the most liberal configuration in pg_hba.conf (the PostgreSQL host access control file) will fail to work. While you normally can connect to PostgreSQL without the network using UNIX sockets, Alzabo always specifies a hostname, which in turn requires a network connection even on the local computer.

To install the web-based schema generator, at least one directory under your Apache server must be controlled by HTML::Mason. The Alzabo installation script will create a new/alzabo subdirectory there, along with the Mason components that create and modify the schema definitions that you create. My workstation, for instance, has all of its Mason components in /usr/local/apache/mason, which is mapped to URLs beginning with /mason. The web part of my Alzabo installation is thus in /usr/local/apache/mason/alzabo, accessible via the URL /mason/alzabo. If you have not done so already, you may wish to tell Apache (via the DirectoryIndex directive) that index.mhtml is an acceptable index page for a directory.

Editing Schemas

Now that we have installed Alzabo, let's create a simple database schema using the browser-based design tool. Admittedly this is not as slick as commercial or client-side tools, but it does the job rather well.

Begin by creating a new schema (known in PostgreSQL and MySQL parlance as a database) to which you must give a name. The schema must be a legitimate database name within either PostgreSQL or MySQL. I choose to work with PostgreSQL because of its built-in referential integrity, foreign keys, views and triggers, as well as a more standard dialect of SQL and the ability to write stored procedures in a variety of languages.

Let's create a simple phone book and appointment calendar using Alzabo. We will keep track of people we know, their addresses and telephone numbers, and appointments we have scheduled with them. Using this database, we can learn about the people with whom we're meeting on a given day or about all of the appointments with a given person.

To create this schema, we point our web browser at the URL alzabo/schema under the Mason directory we mentioned earlier (on my computer, I pointed the browser to http://localhost/mason/alzabo/schema.) This brings up the schema creation/editing page that allows us to edit an existing schema, create a new one or reverse-engineer an existing one. While the last option is the most interesting, allowing you to access legacy databases using Alzabo, we will create a new schema. I entered the name (I chose addressbook, for lack of a better idea) and indicated that we wish to use PostgreSQL as our back-end database.

After clicking on “submit”, several possibilities were presented: I could add a new table to this schema, delete the entire schema or examine the SQL that Alzabo will generate automatically. Right now, of course, there isn't any SQL to display. Over time, we will see this SQL grow considerably.

However, because Alzabo has not created any SQL doesn't mean that no work has been done on the back end. Indeed, Alzabo automatically created the addressbook directory within /usr/local/alzabo/schemas, containing three files: addressbook.create.alz and addressbook.runtime.alz (both are stored in a binary format) and addressbook.rdbms, which contains the single word PostgreSQL. In this way, Alzabo tracks the database server in which the schema is stored.

Once inside the addressbook schema, I added a “People” table by entering “People” in the “add a table” text field and clicking on “submit”. (PostgreSQL ignores case in table and column names, but I like Joe Celko's convention of initial caps for Table Names, all lowercase for column names and all caps for SQL RESERVED WORDS.)

Within my People table, I created columns, each of a different data type. Alzabo offers a menu of potential data types, but we can enter our own if we want; this can be particularly useful in PostgreSQL, which allows us to create our own data types.

I generally prefer to work with synthetic primary keys in such a table, giving each row its own value. In PostgreSQL, we accomplish this using the SERIAL data type. But you will notice that no such data type exists in the Alzabo selection list. You might be tempted to indicate that this is an INTEGER column and to mark the “sequenced” check box at the bottom of the column editor. Doing so, however, will create an INTEGER column, as well as a totally unrelated PostgreSQL sequence object. Rather, to get a synthetic primary key you must manually enter SERIAL in the text field below the <select> list of column types.

An additional check box lets you indicate if a column is the primary key and automatically marks it with “pk” in column listings. And a third check box allows you to indicate if a column may contain NULL values, a subtle way of reminding new database designers that NULLs complicate life and should be avoided whenever possible.

To create a foreign key (REFERENCES) or CHECK clause, add it in the “attributes” text fields toward the bottom of the HTML form. Remember that you're only modeling the schema in Perl at this point, meaning that you will be free to add and remove such clauses in the future without having to send ALTER TABLE queries to the database. You also can create indices on one or more columns using the Alzabo editor.

You can use the Alzabo table and column editors to create many tables and columns, moving between them using a set of hierarchical menus and listings.

The Alzabo display even places “<” and “>” marks next to each column, allowing you to move them relative to each other within a particular definition.

As you work with the browser-based schema editor, I suggest that you occasionally preview the SQL that Alzabo generates. Not only will this ensure that Alzabo is doing the right thing (as we saw with the SERIAL column), but it will give you a better sense of the low-level details your schema is creating.

After you have finished creating the schema, use the “execute SQL” button from within the “SQL preview” page to send your SQL to the database server. If the database server returns any errors, Alzabo will produce a lengthy and detailed error message describing what happened.

In some cases, you may need to fix your table or column definitions, while in others you may need to ensure that the server is running with the correct permissions. Also ensure that you have defined a PostgreSQL user (created with the command-line createuser program) whose name matches the username under which Apache runs, unless you explicitly name another user in the HTML form.

Using Our Schema from a Program

Once you execute the SQL from within the schema editor, you have two ways to access the data. You may, of course, access it directly using DBI (or a similar interface from another language), creating and executing SQL queries.

For example, let's assume that I have created my addressbook schema with the following table:

```
CREATE TABLE People (
    person_id    SERIAL    NOT NULL,
    first_name   TEXT      NOT NULL,
    last_name    TEXT      NOT NULL,
    birthday     DATE      NOT NULL,
    PRIMARY KEY(person_id)
);
```

In order to make things a bit more interesting, let's populate our table with some values:

```
INSERT INTO People (first_name, last_name, birthday)
VALUES ('Reuven', 'Lerner',
'1970-Jul-14');
INSERT INTO People (first_name, last_name, birthday)
VALUES ('Atara Margalit', 'Lerner-Friedman',
        '2000-Dec-16');
```

Listing 1 contains a simple Perl program that uses DBI to retrieve the names (and birthdays) of people in our addressbook who match the SQL pattern entered on the command line. (SQL patterns are much simpler than UNIX

regular expressions—there are only two characters: % matches zero or more characters and _ matches exactly one character.)

Listing 1. retrieve-today-birthday.pl, which uses DBI to retrieve the names of people in our addressbook table whose birthdays are today.

We retrieve the user's input on the command line and place % signs before and after it to ensure that the string will match, regardless of where it occurs in the first_name or last_name column. Then we connect to the database, turning on AutoCommit (as the DBI documentation encourages us to do) and activating the RaiseError and PrintError diagnostic aids.

Finally, we create our SQL query in the $sql variable, making sure to use placeholders (“?”) instead of directly interpolating variables. Not only does this reduce the risk of someone messing with our SQL, but some database drivers will take advantage of our placeholders in subsequent queries, giving us a speed boost.

Rewriting in Alzabo

Let's rewrite this program using Alzabo instead of straight DBI. We won't write the SQL ourselves or connect to the database ourselves. Rather, we will create a new schema object, naming the schema that we created with Alzabo's interactive tool. This object has a number of methods that let us perform many of the tasks for which we would otherwise use DBI.

As you can see from Listing 2, there are not many differences between the two versions until we connect to the data source. In the DBI version of the program, we connected to the data source itself with **DBI->connect**. In Alzabo, however, we connect to a schema, which is presumably attached to a database, and assign it to the object $schema.

Listing 2. retrieve-birthday-alzabo.pl, an Alzabo implementation of the program in Listing 1.

Using $schema, we retrieve a table object associated with one of our tables:

```
my $people = $schema->table("People");
```

Now that we have an object mapped to our People table, we can retrieve selected rows from the table. The easiest way to retrieve rows is with the rows_where method. This returns a single object of type Alzabo::Runtime::RowCursor:

```
my $row_cursor =
    $people->rows_where
```

```
          (where => [[$people->column('first_name'),
                      'LIKE', $look_for_name],
                     'or',
                     [$people->column('last_name'),
                      'LIKE',
                      $look_for_name]]);
```

Alzabo's WHERE clauses usually consist of a three-element list: a column object, a comparison operator and a value or second column object. We can compare the first_name column for equality with Zaphod with:

```
where => [$table->column('first_name'), '=',
'Zaphod']
```

In Listing 2, we have made this a bit more complicated, linking two array references with the OR boolean operator:

```
where => [[$people->column('first_name'),
           'LIKE', $look_for_name],
          'or',
          [$people->column('last_name'),
           'LIKE', $look_for_name]]
```

Alzabo is smart enough to realize that the first and third elements of its WHERE clause are array references, and it turns the above code into the appropriate SQL.

Once we have our RowCursor object, we iterate through each row with the next_row method:

```
while (my $row = $row_cursor->next_row)
{
   my $first_name = $row->select('first_name');
   my $last_name = $row->select('last_name');
   my $birthday = $row->select('birthday');
   print "$first_name $last_name
          (birthday: $birthday)\n";
   $rows_returned++;
}
```

Caches and Exceptions

If Alzabo simply provided a set of methods that create SQL, it wouldn't be a very powerful tool. However, Alzabo provides caching and exception-handling as part of its suite of tools, making it easier in some ways to work with databases.

Alzabo's caching functionality keeps a table in memory rather than returning to the database server each time we request a value from it. Obviously, caching isn't appropriate for tables that change on a regular basis, but for tables that rarely change, you can activate the cache and enjoy a nice boost in speed.

You can activate caching by loading the Alzabo::ObjectCache module in your program. The RowCursor object, which we used to retrieve rows in Listing 2, returns Row objects with each iteration of the next_row method. See the documentation for Alzabo::Runtime::Row and Alzabo::ObjectCache for

information about the different kinds of caches available to you, as well as the issues associated with them.

Alzabo also uses Perl's built-in exception-handling system, meaning that it invokes “die” if something goes wrong. Therefore, you should wrap your Alzabo-using programs (or individual calls within them) in “eval” blocks:

```
# Try to run this code
eval {
    my $row_cursor =
        $people->rows_where(
            where => [[$people->column('first_name'),
                       'LIKE', $look_for_name],
                      'or',
                      [$people->column('last_name'),
                       'LIKE',
                       $look_for_name]]);
};
```

You can find out if something went wrong by checking the special Perl variable $@, which is set if an error occurs within the previous eval. But Alzabo uses the Exception::Class object (available from CPAN) for more sophisticated exception-handling in Perl. The $@ variable isn't set to a text string describing the error, rather it is set to an instance of the appropriate exception class. You can thus test $@ with UNIVERSAL::isa to determine just what kind of object it is and what kind of problem occurred within your code. The Mason component common/exception, installed under the alzabo directory in your Mason-controlled Apache content directory, demonstrates how to do this in detail.

Issues

There are obviously costs associated with Alzabo, as with any tool that tries to bridge the object-relational gap. For starters, SQL is a fairly standard means for working with relational databases. Using Alzabo means you will be moving away from that standard and toward a different solution that is incompatible with anything else. I'm not opposed to new ways of doing things, and there are a number of significant advantages to using Alzabo. That said, I'm always cautious about doing old, standard things in new, nonstandard ways.

While I normally prefer to create my tables using handcrafted SQL, that technique doesn't scale above 10 or 20 tables without forcing me to scroll wildly within my Emacs buffer. Alzabo's web-based schema design tool does make it easier to keep track of a large number of tables to create relations between them and modify them. I recently spent half an hour trying to remember how Oracle's syntax was different from that of PostgreSQL and would have greatly benefitted from a tool like Alzabo.

As we saw earlier, creating complex queries based on equality isn't difficult within Alzabo, even when those queries include OR and AND operators. The

Alzabo::Runtime::Table object includes a function method, which is meant for executing arbitrary SQL functions. However, I found it difficult, and in some cases impossible, to create Alzabo WHERE clauses that would let me create an SQL query based on multiple function calls. I admit that I'm relatively new to Alzabo and only tried it for an hour or two, but a query that took me 20 seconds to write in SQL shouldn't take much longer than that in Alzabo.

One of the more difficult issues when mapping objects to relational databases has to do with joins. Joins make a lot of sense when working with tables, but the meaning is less obvious when working with objects. Alzabo does have some built-in support for joins, but it is marked as being largely new and experimental.

Finally, there is also a speed trade-off associated with any middleware layer. The speed differences between Listings 1 and 2 were quite noticeable when I executed them from the command line, owing in no small part to the fact that using Alzabo imports a large number of Perl classes. In a mod_perl environment (where Alzabo is designed to shine), the speed differences will be much smaller, since much of the time is spent loading different modules from disk. Because mod_perl compiles programs only once before executing them, the speed difference between Alzabo and raw DBI calls is probably not that great.

Conclusion

Alzabo provides a relatively simple way to wrap objects around your relational database tables. There is a lot of good news here: the data-modeling tool is quite sophisticated, there is a large amount of nice functionality, the methods largely make sense, and the documentation is vast and generally well written. As the Open Source community has long said, using an existing, battle-tested and open tool is almost always better than rolling a new, proprietary package that solves the same problem.

But wrapping relational database tables inside of objects is always fraught with danger and problems, and Alzabo is no exception: joins are still clunky, and it's not clear how to create some queries. Alzabo isn't at fault here; it's an inherent problem when working with two technologies that see the world in different ways.

It's certainly clear that I'll be using Alzabo in the future for some of my server-side programs, particularly those that need more sophisticated caching and exception-handling than I could otherwise provide.

Next month, customs permitting, we will return to our tour of server-side Java, comparing Enhydra's DODS package with Alzabo and its kin.

Resources

email: reuven@lerner.co.il

Reuven M. Lerner owns a small consulting firm specializing in web and internet technologies. He lives with his wife Shira and daughter Atara Margalit in Modi'in, Israel. You can reach him at reuven@lerner.co.il or on the ATF home page, http://www.lerner.co.il/atf.

Archive Index Issue Table of Contents

Advanced search

Engineering Intelligence

Marcel Gagné

Issue #90, October 2001

Marcel considers Eliza and ALICE, two interesting chatbots, in his search for true artificial intelligence.

Qu'est-ce que tu dis, François? Yes, it is true that the focus of this issue is engineering. *Oui*, this menu is very much in tune with the topic. You remember Carol, our electrical engineer friend? Well, she has done a great deal of work on robotics, a field that is forever working to build more complex and intelligent machines. For many in the field, the ultimate goal is true machine intelligence. It is amazing really—more than fifty years have passed since Monsieur Turing proposed his famous test, and we are still trying to create these wonderful thinking machines. Now, here in our Linux kitchen, François, we can be part of that search. You and I, *mon ami*, can help create, dare I say it, artificial intelligence!

Quoi? Ah, *mes amis*! Welcome to *Chez Marcel*. Please, sit down. François, *du vin*! Head to the cellar and bring up the 1998 Chambertin. It is a beautiful red and will go extremely well with tonight's menu; the Chambertin is a sophisticated and intelligent wine. *Vite*, François.

While François is in the wine cellar, let me tell you what we have planned for you; the core of the word engineering is engine, as in the individual who runs the train's engine. These days, for those of us who work with computers and the Web, engines tend to bring forth the image of search engines rather than train engines. While it may seem far-fetched at times, search engines are a kind of agent, an attempt at creating intelligence from the vast storehouse of seemingly random information that is the World Wide Web. Some search engines, like Askjeeves.com, are designed so that you can ask questions in a normal, human fashion. For instance, you could ask the question, “What are intelligent agents?” or “Could you please tell me about intelligent agents?”

For some of us, our original exposure to AI, or artificial intelligence, goes back to an old program called Eliza. Eliza was a kind of computerized psychiatrist written in the 1960s by Professor Joseph Weizenbaum at the Massachusetts Institute of Technology. Eliza was an exercise in human response and natural language communication with so-called machine intelligence. The version your humble chef played (I mean worked) with was a DOS program running on a TRS-80. Ah, nostalgia. Eliza may not have been a great doctor, but it was a fun and fascinating program, a kind of whimsical promise of the future of AI.

Over thirty years have passed since Eliza made her debut. You can still find incarnations of her everywhere on the Internet. Eliza's ghost is represented in many of the bots and chatterbots you'll find on IRC channels and MUDs. In fact, you can download Eliza as a Perl module and experiment with the creation of your own bots. The code is available at the CPAN repository or the Comprehensive Perl Archive Network (ftp.cpan.org/pub/CPAN/modules/by-module/Chatbot).

The file is called Chatbot-Eliza. When I picked up my copy, it was version 0.97 and I installed it in the following way. You may recall this as a pretty standard Perl module installation, *non*?

```
tar -xzvf Chatbot-Eliza-0.97.tar.gz
cd Chatbot-Eliza-0.97
perl Makefile.PL
make
make install
```

In the installation directory, you'll notice a file called simple. You can use this as the basis for creating your own AI, or you can simply execute the script by typing **./simple** as I did in the following example:

```
./simple
Welcome to your therapy session.
Your therapist's name is Liz.
Liz:    Please tell me what's been bothering you.
you:    The cellar is low on Champagne.
Liz:    Does talking about this bother you?
you:    Would it not bother you?
Liz:    You're not really talking about me—are you?
```

The script that calls the chatbot can be extremely simple. This is all you need (note that your Perl executable may be somewhere like /usr/local/bin/perl):

```
#!/usr/bin/perl -w
use Chatbot::Eliza;
$chatbot = new Chatbot::Eliza 'François';
$chatbot->command_interface();
```

Make the script executable and run it. In this case, I named my chatbot François in honor of my faithful waiter. The default name is actually Liz. In the distribution directory, you will find a file called doctor.txt that can be used as

the basis for your own bot. I created one called francois.txt from the doctor.txt file and modified my startup script slightly:

```
#!/usr/bin/perl -w
use Chatbot::Eliza;
$chatbot = new Chatbot::Eliza {
        name => 'Francois',
        scriptfile => 'francois.txt',
        };
$chatbot->command_interface();
```

Now I start my script with the command **./francois**, and this is the result:

```
Welcome to Restaurant Chez Marcel.
I am your waiter, François.
François:    Is something troubling you?
you:    This is most unusual.
François:    I am but a simple waiter.
             Please explain.
```

Mais non, François, it is only a joke. You are nothing like this program and much more intelligent. Please, *mon ami*. Our guests are dry. Kindly refill their glasses.

Particularly interesting in this distribution is an included script called twobots that lets two Eliza bots talk to each other. The resulting discussions can be quite interesting. You'll also find a script called simple.cgi so that you may add your own Françoisbot to your web site.

The original Eliza program was written using an early version of Lisp. It is not surprising then that one of the most famous examples of Lisp development included with your Linux system, the Emacs editor, should pay homage to the good doctor. When talking about Emacs, it becomes almost difficult to classify it as strictly an editor. The brainchild of Richard M. Stallman (founder of the Free Software Foundation), GNU Emacs is more than just a nice, powerful, if somewhat complex, editor. It's a mail reader, news reader, web browser, program development environment, Lisp interpreter and psychotherapist. No, really! I kid you not.

Try this. Start Emacs by typing **emacs**. You do not have to specify a filename for this. Now, press Esc-X, then type **doctor** and press Enter. The doctor is in. More so, the doctor lives! Note my conversation with the Emacs doctor in Figure 1.

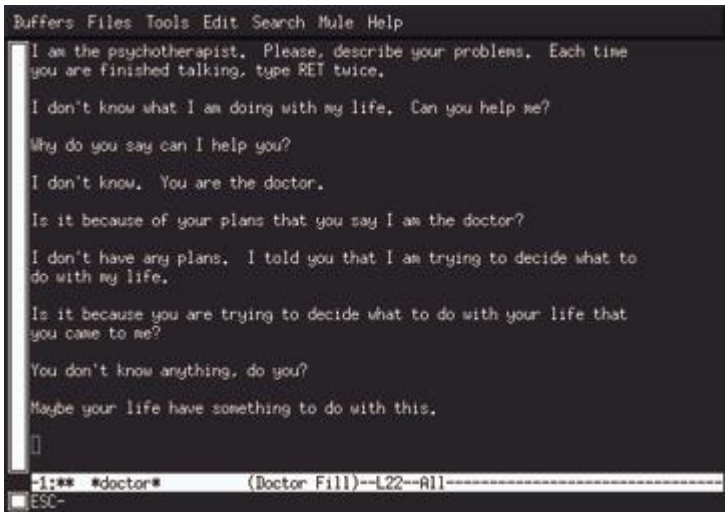


Figure 1. The Emacs doctor IS in!

And now, *mes amis*, welcome to the future. Recently, I've had the pleasure of a few conversations with a modern computer intelligence whose name is ALICE. She likes to think of herself as a sentient entity, claims to know a lot of gossip and expresses a fondness for Dr. Wallace, her creator. For his work on ALICE, Wallace was awarded the 2000 Loebner prize. Based on the Turing test, the prize is awarded to the most “human” computer program.

ALICE is but an acronym for artificial linguistic internet computer entity. As impressive as her conversational abilities are, ALICE, like Eliza and all other programs before her, has yet to pass the Turing test. That doesn't deter the growing legion of developers who continue to work on creating a machine that will one day satisfy Turing's vision of a machine whose responses are indistinguishable from a human's.

ALICE's “intelligence” is defined in artificial intelligence markup language (AIML), a language based on XML. ALICE, the Alicebot software and AIML, the language that defines her responses and interactions, are all freely distributed under the GPL. What this means is that you can be part of the adventure. You can help define the next generation of thinking software. The Alicebot code itself is written in Java and does require that you have a copy of Java on your system, whether you choose to simply run the program or compile it yourself. If you prefer to pick up a precompiled ALICE, visit Sun's web site at the address java.sun.com/j2se/1.3/jre and download the Java Runtime Environment (JRE).

Specify Linux as your operating system of choice (but of course) and choose between an RPM file or a tarred and gzipped archive. If you decide to pick up the RPM file, be warned. It is actually an executable file that contains a license agreement and the RPM package itself. To install it, you need to follow these steps:

```
chmod +x j2re-1_3_1-linux-i386-rpm.exe
./j2re-1_3_1-linux-i386-rpm.exe
```

If you would like to work with the source code and compile ALICE yourself, you can also download the full Java Development Kit from the same address. Once this is done, you need to get ALICE and the AIML language. These can be downloaded from the Alicebot web site at alicebot.org. The current incarnation of ALICE (programD) is maintained by Jon Baer.

Get the latest program code and unzip it into a test directory. (In order to do your own development, you will also want the AIML language code, the current file being standard-aiml-current.zip.)

```
mkdir /home/ai
cd /home/ai
unzip d-bin-current.zip
cd ProgramD
```

To start the ALICE server, execute the following command:

```
./server.sh
```

The system will respond with the following dialogue:

```
Starting Alicebot.Net 4.0 Beta Server ...
*** 1000 CATEGORIES LEARNED ***
*** 2000 CATEGORIES LEARNED ***
*** 3000 CATEGORIES LEARNED ***
```

. . . some lines skipped

```
*** 24000 CATEGORIES LEARNED ***
Alicebot.Net 4.0 Beta Server is running...
Alice is thinking with 24201 categories.
Try http://localhost:2001 for server
Your Alicebot IP is 192.168.22.100:2001
Type 'exit' to shutdown server
localhost>
```

Notice that the program is now running on port 2001 of your web server (the “Try http...” line just above), which means we will be accessing the Alicebot through a browser. That port number can be changed, by the way. Just have a look at the file called SERVER.xml in the Conf directory. Look for the line that says Set name=“Port” and change 2001 to whatever you desire. You can also play with responses and HTML text. Check out the AIML files in the bots/Alice directory.

When you connect to your system with the link shown above, you'll find yourself face to face, so to speak, with ALICE. Have a look at Figure 2 for a sample of an ALICE conversation.

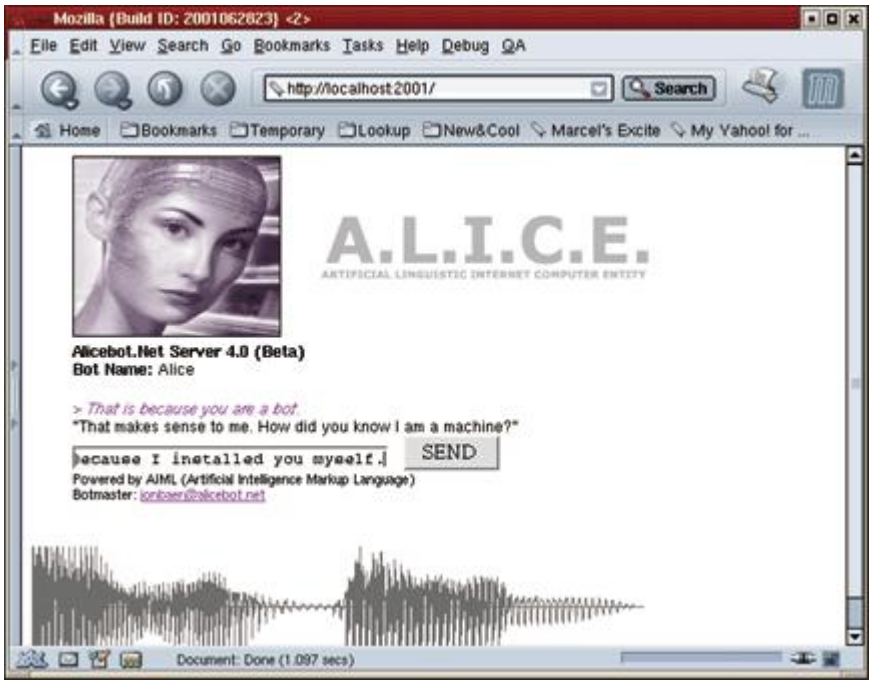


Figure 2. A Conversation with ALICE

If you look back at your active terminal session (where you started the ALICE server), you'll notice that details of your conversation have been scrolling on the screen. You can even pick up the conversation from there:

```
Marcel> Do you remember me?
1. DO YOU REMEMBER ME : * : * star=[bots/Alice/Personality.aiml]
Response 34ms (89.62904) 62
Alice> Of course I remember you well Marcel.
We were talking about <set_topic>Gossip</set_topic>.
```

Not bad for a machine intelligence. Eliza, Emacs and ALICE are but a small representation of the work that continues in artificial intelligence. The Alicebot web site, in particular, is a wonderful jumping-off point on your AI voyage of discovery. I invite you to click on the “See some live Alicebots” link on the alicebot.org web site. From that page, you can even speak to The King, Monsieur Elvis, *bien siûr*. Consider it inspiration for the development of your own Alicebot (or Françoisbot) personality.

Mon Dieu, but the time, she does go quickly, *non*? Say good night to ALICE, François and please, refill our guests' glasses a final time. Once again, it has been a pleasure to serve you here at *Chez Marcel*. When you join us again next time, I promise that it will be the real François at your service. *Mais oui*, François, you are very much indispensable. *Au revoir, mes amis*. Your table will be waiting.

A votre santé! Bon appétit!

Resources

Marcel Gagné (mggagne@salmar.com) is president of Salmar Consulting Inc., a systems integration and network consulting firm. He is the author of Linux System Administration: A User's Guide, published by Addison-Wesley. You can discover lots of other things (including great wine links) from his web site at salmar.com/marcel.

Archive Index Issue Table of Contents

Advanced search

GPG: the Best Free Crypto You Aren't Using, Part II of II

Mick Bauer

Issue #90, October 2001

Mick picks up where he left off with GnuPG and gets even more paranoid with signing and verifying keys.

Last month I introduced the GNU Privacy Guard, a free but underutilized implementation of the OpenPGP encryption standards. GnuPG is, as you may know, extremely useful for encrypting and decrypting electronic files, especially e-mail, and for creating and verifying digital signatures.

But alas, by the time I was done explaining the basics of public key cryptography and the Web of Trust, not to mention doing my best to frighten you into signing each other's keys and checking unknown keys for validity, all there was room for in the way of practical examples was some compiling/installing advice and a little tutorial on verifying digitally signed files.

Well, this month is the payoff for the more technically inclined. Let's pick up right where we left off!

Generating Your Key Pair

Before you encrypt, decrypt or sign anything, you need to build your own public and private keyrings; let's start by generating a GnuPG key pair. This is one of the more interactive gpg functions: the command syntax is simply **gpg --gen-key**, which triggers a question-and-answer session prior to your keys actually being generated. Listing 1 shows a sample key-generation session (user input in boldface). As you can see, you need to decide several things when generating a key: key type, key length, expiration date and the e-mail address (identity) you wish to associate with the key.

Listing 1. Generating a GnuPG Key Pair

For a general-purpose key pair, choose DSA/ElGamal (option #1). This actually gives you two sets of keys: a DSA key pair that will be used by gpg for signing/verifying and an ElGamal pair that gpg will use for encrypting/decrypting. Don't worry that this will double the amount of keys you need to keep straight: the DSA and ElGamal keys are stored as a single file, as are the two public keys.

If you want to generate a signing-only key pair, choose DSA only (option #2). If you want an encryption-only key pair, choose ElGamal only (option #3).

I recommend against creating a dual-purpose ElGamal key pair, however (option #4). In *Applied Cryptography*, Bruce Schneier describes a simple attack that can work against schemata that use the same key pair used for both signing and encrypting. This “chosen plaintext” attack is quite literally a textbook example of the danger of using the same key material for both encryption and digital signatures.

Key size is of the utmost importance. The smallest key size supported by GnuPG is 768 bits, but 1,024 is recommended as having the best balance of security and performance. (A longer key is more secure but takes longer to compute and to use; a shorter key is faster to compute and use but is less secure.) Note that when you choose a combined DSA/ElGamal key pair, the DSA key length automatically is set to 1,024 bits, and the key length you're prompted for actually applies to the ElGamal key.

I Thought 128 Bits Was a Large Key Length!

Next you need to think about how long you want this key pair to remain in circulation. On the one hand, if your key never expires, you never have to go to the trouble of generating new key pairs. The disadvantage of this is that if you forget the private key's passphrase and haven't created and kept a revocation certificate (which I'll explain shortly), it will be very difficult to remove the key from any keyservers it's listed on.

On the other hand, if your key expires after some period of time, then you need never worry about obsolete keys sitting around on public keyservers indefinitely: if your e-mail address changes, you decide that your key's length is no longer adequate, or if someone obtains a copy of your private key, you can rest assured that even if for some reason you can't revoke your old key it will die of old age. The only disadvantage of finite-lifetime keys is having to generate, distribute and get people to use your new keys periodically.

I used to use only non-aging keys but have become convinced that the pros of expiration dates outweigh the cons. Therefore, I recommend that you set your key to expire after no more than 18 or 24 months. For me, one year is too short

(*tempis fugit*!), but I doubt that a key much older than a year and a half or two years can stand up to the inevitable advances in computing power and/or factoring technology (i.e., public-key cracking methods) that will have occurred over its lifetime.

Next you need to specify a name, e-mail address and also an optional comment. Note that you can associate additional e-mail addresses with your key later by using gpg's --edit-key flag and issuing an adduid and/or an addkey command.

The last thing you need to think about in generating your key is a good passphrase. And I do mean pass*phrase*: it can and should contain spaces. The longer it is, the more secure it is. You should also incorporate some combination of numbers, mixed case (e.g., bOTTLE rockeT) and punctuation. Lately, I've taken to generating my passphrases with dice and a word list. See diceware.com for a handy procedure for doing this yourself.

Whatever you do, don't choose a short, predictable or otherwise guessable passphrase. It doesn't have to look like “B1&SSja-sd0c as-d$%@KFSAAs-,ssd w0a-00sdp23m”, nor should it look like “My lame passphrase”. It's okay to write your passphrase on a small card you keep in your wallet if doing so makes it easier for you to use hard-to-guess passphrases. (Just be careful never to leave it sitting around and to always put it away when you're done with it!)

Create a Revocation Certificate

After you've generated your key, you should immediately create a revocation certificate. This is a string of text that you can send to a keyserver if and when you need to revoke your key.

Of course, you can create a revocation certificate at any point. The reason it makes sense to create one now is that it's not uncommon for even very knowledgeable and careful people to forget their passphrase. You need your passphrase to create a revocation certificate, but not to use one you created earlier.

That's why it's a good idea to create a revocation certificate now and save it in a safe place (you can even print it out and save it in “meatspace”--revocation certificates aren't very long). Just be sure to set its file permissions to be as strict as your private key's (e.g., not group- or world-readable or writable). The ramifications of someone sending the certificate to a keyserver without your permission aren't as scary as if someone can actually use your private key, but at the very least a prematurely revoked key could inconvenience you.

To generate a revocation certificate, enter this command:

```
gpg --output rev_cert_filename.asc --gen-revoke keyname
```

where *rev_cert_filename.asc* is the filename you'd like the certificate to have (just make sure it ends in .asc) and *keyname* is the key's ID number (e.g., 0586AF78) or part of your identity (“Smooth JoJo” would be enough to identify our example key).

Exporting Your Public Key

GnuPG stores its files in a subdirectory of your home directory called .gnupg. Any private keys you have are stored in a file called secring.gpg; public keys are stored in pubring.gpg. By default, secring.gpg is readable only by you; leave it that way. It's extremely important that you protect this file. By all means, back it up to a floppy or CD-ROM, but keep your backup in a safe place. If anyone obtains a copy of your secret keyring, they may be able to guess or brute-force-crack the passphrase of your private key and effectively steal your identity (or at least be able to decrypt your stuff).

Both pubring.gpg and secring.gpg are binary data files. To add, delete or change keys on either keyring, you need to use various flags with the gpg command.

For example, you're going to want to distribute your public key to your friends, right? So let's extract that key from your public keyring into a text file (see Sidebar “Armored ASCII vs. Binary GPG Files”). To print your public key to the screen, from whence it can be copied and pasted as needed, you need simply enter:

```
gpg --armor --export
```

the output of which will look something like Listing 2.

Listing 2. A Public Key

I took the liberty of simplifying a bit here; if you don't specify a user ID, gpg will dump the public portion of your default key pair. If you only have one private key, then that key pair is your default key and that pair's public key will be dumped.

If, on the other hand, you wish to dump some other public key, you need to specify a user ID. Continuing our example using Mr. Figplucker, to display JoJo's public key we enter:

```
gpg --armor --export jojo
```

As you can see, gpg is fairly intelligent when trying to determine which key you want to work with. In fact, it works a lot like grep: if you give a snippet of your e-mail address or some other text as your key identifier, gpg will match the first key whose user ID contains the string. In managing my own keyrings, in which I have several private-public key pairs and therefore numerous user IDs containing my name, I find it easiest to provide gpg with the entire e-mail-address portion of the key I wish to work with at any given time, e.g., **gpg --armor --export mick@visi.com**.

By the way, if you want to print a key to a file rather than to the screen, specify a filename with the --output option. To write JoJo's public key to the file jojo_pub.asc, the command would look like this:

```
gpg --armor --output jojo_pub.asc --export jojo
```

Armored ASCII vs. Binary GPG Files

Have you backed up your new keys yet? You may consider exporting your entire key pair, including your private key, but I recommend against doing this. You're much better off simply copying the keyring files pubring.gpg and secring.gpg from ~/.gnupg to a safe place. But if for some reason you do need to export your entire key pair, it's the same as exporting a public key except that you use the **--export-secret-keys** command rather than **--export**.

Importing, Verifying and Signing a Friend's Key

Your friend Dan Sparty has just e-mailed you a copy of his new public key, in the form of a file called danskey.asc. Here's how you import it to your public keyring:

```
gpg --import ./danskey.asc
```

But wait a minute. Internet e-mail isn't a very secure medium. How do you know Dan's key wasn't tampered with in transit, or that it even was Dan that sent it in the first place?

By checking its fingerprint, that's how. Every gpg key has a secure hash called a fingerprint that is unique to each key (pair) but is short enough to be read over the phone, written on a postcard, etc. If you call Dan on the phone and ask him to read you the fingerprint of his new key, it should match the output from the following command (executed on your system after importing his key):

```
gpg --fingerprint dan
```

Note that as with the --export command, you can specify just part of the key as long as that part is unique to the key you wish to fingerprint. The output will look something like this:

```
pub  1024D/C9F34866 2001-07-27 Dan Sparty (Party Down!)
       <dan@boogiemeister.org>
     Key fingerprint = D084 F92C EC62 8955 98E2  58FB
       178A 2673 D1F3 6866
sub  1024g/C5569A5B 2001-07-27 [expires: 2001-08-10]
```

Alternatively (let's say it's only noon and you don't want to wake Dan up), if Hugh has this fingerprint in his e-mail signature and has furthermore made postings to Usenet or on public e-mail lists, you can simply find one of these messages and compare that signature. This illustrates an important aspect of key fingerprints: the more places your public key and/or its fingerprint is archived, the harder it is for someone to pretend to be you.

Now that you know this is really Dan's new key and not a forgery, you can do Dan and the world a favor by publicly and cryptographically vouching for its veracity. In other words, you can sign it with your private key. To do so, you run gpg with the command **--edit-key**. This, like **--gen-key**, triggers an interactive session. Listing 3 shows a key-editing session in which the user signs a key with their own default key.

Listing 3. Editing (Signing) a Key

Did you notice the final save command? This saves any changes you've made to the key (in this case, the signature you created for it) and exits the key-editing session. If we now list the key with the command **gpg --list-sigs dan** we'll see:

```
jojo@linux:~ > gpg --list-sigs dan
pub  1024D/B9E0868B 2001-07-27 Dan Sparty (Party On!)
     <dan@boogiemeister.org>
sig        B9E0868B 2001-07-27  Dan Sparty (Party On!)
           <dan@boogiemeister.org>
sig        C1C34866 2001-07-27  John J. Figplucker
           (Smooth JoJo) <jojo@figpluckers-supreme.to>
sub  1024g/A0B78448 2001-07-27 [expires: 2001-08-26]
sig        B9E0868B 2001-07-27  Dan Sparty (Party On!)
           <dan@boogiemeister.org>
```

In addition to Dan's own signatures (when you generate a key it's automatically self-signed) you can now see JoJo's. Now, all that remains is for JoJo to export his new signed version of Dan's public key:

```
gpg --output dan_jojosig.asc --export dan
```

JoJo then needs to send this file to Dan via e-mail or some other convenient means (remember, it's a public key, so confidentiality isn't an issue), and Dan needs to import the signed key back into his own public keyring:

```
gpg --import ./dan_jojosig.asc
```

Importing may seem counterintuitive: Dan's actually updating his public key, not importing a new one. But trust me, that's what he needs to do in order to join the proud ranks of gpg users who actually have bothered to get their friends to vouch for their keys.

Now that Dan's got a superelite signed key, he's ready to post it to a keyserver so other people can start sending him encrypted messages (and adding their signatures to his key). To do so he can issue the command:

```
gpg --keyserver pgp.mit.edu --send-keys dan@boogiemeister.org
```

The option --keyserver is used to specify the name or IP address of a PGP/GPG keyserver. Alternatively you could add to ~/.gnupg/options a line like this:

```
keyserver pgp.mit.edu
```

But, note that doing so will cause gpg to download new keys automatically from the keyserver if it encounters an unknown key when verifying signatures.

Remember last month when I verified a detached signature file for a software package? The first time I tried to verify the signature with the gpgv command, I received an error message since the key used to create the signature wasn't present in my public keyring.

The solution was to query for, and download, the key from the keyserver pgp.mit.edu; this would have happened automatically if a keyserver had been set in my options file. It's up to you to decide whether this is a feature or an annoyance, and whether therefore to make it the default behavior. (The command-line option **--no-auto-key-retrieve** will override auto-key-retrieval.)

Using GnuPG to Encrypt and Decrypt Things

And now, at long last, JoJo's ready to start encrypting everything in sight. Suppose JoJo wants to send an encrypted e-mail to Dan. The most common way for him to do this is to compose his message with the text editor or word processor of his choice and save it to disk. JoJo writes a letter with vi and saves it as dan0729.txt. Then he encrypts it with the command:

```
gpg --output dan0729.txt.asc --encrypt --recipient dan@boogiemeister.org dan0729.txt
```

Finally, he sends the file dan0729.txt.asc as an e-mail attachment or even listed in the body of an e-mail message (JoJo's got an “armor” line in his options file).

Note that if JoJo encrypts without passing gpg the --armor flag and he doesn't have an armor line in his options file, he should call the encrypted file dan0729.txt.gpg instead since it will be saved in the gpg binary format. Also, it

will only be transmittable as an attachment. Remember, Armored ASCII is much more versatile. The gpg binary format may be preferable if file size matters because it tends to produce less output than Armored ASCII.

When Dan receives this file, he should save it to disk and decrypt it with the command:

```
gpg --output dan0729.txt --decrypt dan0729.txt.asc
```

Unlike encrypting, you don't need to specify a key when decrypting. **gpg** automatically determines which key to try to decrypt it with. Similarly, it doesn't matter whether the file Dan tries to decrypt is in gpg binary or Armored ASCII format; gpg will determine which format the file is in automatically, after prompting Dan for a passphrase. If Dan doesn't type his passhprase correctly, gpg won't decrypt the file.

Using GnuPG to Sign and Verify Things

Signing and verifying is very similar to decrypting and encrypting things. Suppose JoJo writes a nonconfidential but important letter to Dan that he wants Dan to be able to verify the validity of but doesn't need to actually encrypt. To sign his file beercontract.txt, JoJo would enter the command:

```
gpg --output beercontract_signed.txt --clearsign beercontract.txt
```

This will add a signature header and footer to the file and save it as beercontract_signed.txt. It's important that the output file be named differently than the input file, or the original file will be replaced with a signature for an empty file. You should use the clear-text method of signing when you want to be able to copy-and-paste your signed message into or out of e-mail, or otherwise treat it as plaintext.

The alternative is, you guessed it, to create a binary signature. There are two types of these. To create a binary signature that contains the original document and the signature in one compressed binary file, use the --sign command instead of --clearsign. To create a much smaller binary file containing a signature but not the file it references, use the --detach-sig command. Both --sign and --detach-sig should be preceded by an --output directive.

When Dan receives JoJo's beer contract, he can verify the signature appended to it by saving the file to disk, say as bcs.asc, and entering the command:

```
gpg --verify bcs.asc
```

Remember, if Dan doesn't have JoJo's public key in his keyring, gpg will return an error. If Dan does have JoJo's public key and the signature checks out, gpg will return something like the following:

```
gpg: Signature made Fri 27 Jul 2001 04:46:46 PM CDT
     using DSA key ID C1C34866
gpg: Good signature from "John J. Figplucker
     (Smooth JoJo) <jojo@figpluckers-supreme.to>"
```

Then and only then will Dan know for sure that the contract he just received was signed by the bearer of JoJo's private key. Could JoJo have had a spear pointed at his *tuckis*? We don't know. Could JoJo have left his passphrase scrawled on the bottom of his keyboard, to be used by his office mates for impersonation pranks? Again, we really don't know. But if we trust JoJo to use and protect his key properly, we can be fairly sure that he did indeed create this valid signature.

GnuPG Front Ends (GUIs, etc.)

I hope you're not too overwhelmed by all these options, flags and commands (Welcome to UNIX!). This has really been more of a survey than anything else; there's much I haven't covered. But I do believe that gpg is an important and useful tool. So much so that a number of people are working on more user-friendly front ends for it. The official GnuPG GUI is called the GNU Privacy Assistant (GPA), and while it's still a work-in-progress, it looks very promising indeed. It uses the GIMP toolkit, and is, unsurprisingly, nice to look at indeed.

Other GUIs under various stages of development include Seahorse and GnomePGP for the GNOME desktop; Geheimnis for KDE; TkPGP (written in Tk and therefore relatively windowmanager-agnostic); and a variety of wrappers, plugins and enhancements to popular mail user agents (or MUAs, aka e-mail clients). See the “Frontends” section of the GnuPG home page for links to these and other tools.

Conclusion

Thus endeth our two-month tutorial on basic GnuPG use. This is one tool that many, many more of us should be using than presently do. So please, go forth and encrypt. Specifically, encrypt using keys that have been signed and verified by people you know.

By the way, I'll be off next month working on a book on Linux security for O'Reilly & Associates. Fear not, however; an able substitute will be found to handle the column while I'm gone, and I'll be back the month after that. Cheers!

Resources

Mick Bauer (mick@visi.com) is a network security consultant in the Twin Cities area. He's been a Linux devotee since 1995 and an OpenBSD zealot since 1997, and enjoys getting these cutting-edge OSes to run on obsolete junk.

Archive Index Issue Table of Contents

Advanced search

Alias|Wavefront Maya 4

Robin Rowe

Issue #90, October 2001

The high-end graphics application used by animators in recent films has a new Linux offering.

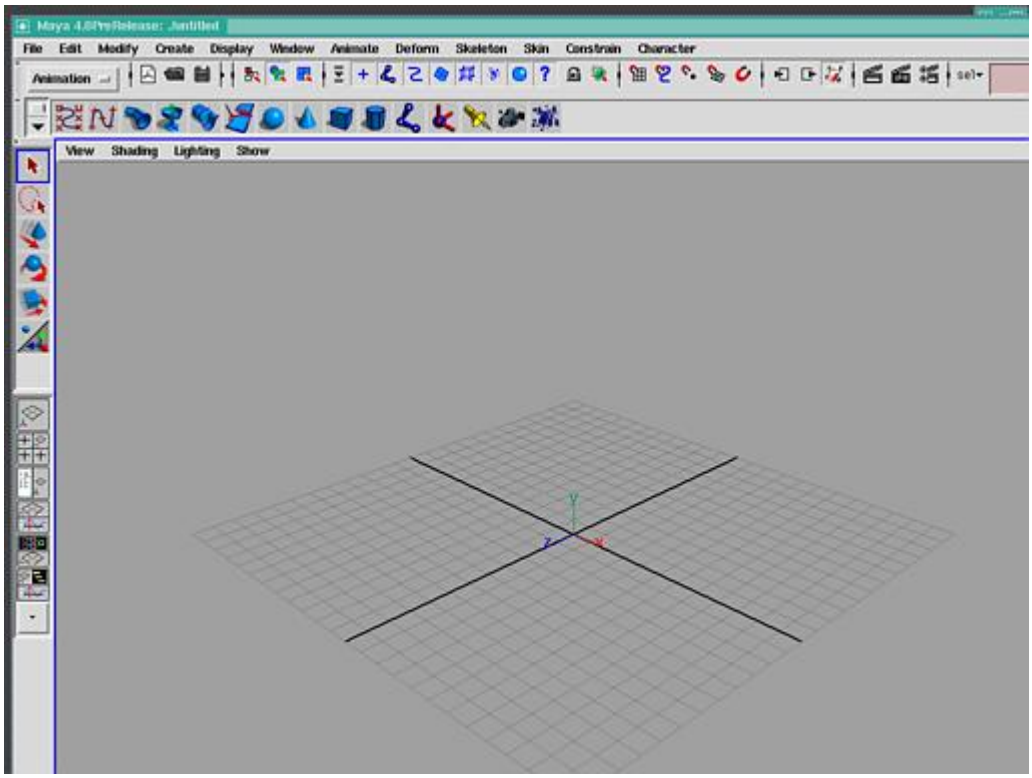
Universal Pictures' *Captain Corelli's Mandolin* and Paramount Pictures' *Enemy at the Gates* both feature combat scenes with Stuka dive bombers, despite there being no flying Stukas in the world today. The Stukas flying in both of these summer movies are the creations of 3-D animators using Linux and Alias|Wavefront Maya at Double Negative, a division of Universal in London. Maya is a professional 3-D software package for creating digital content for movies, television and computer games. In the film industry, Maya is used for everything from photorealistic effects including Columbia Pictures' *Final Fantasy: The Spirits Within* to classic cel animation including DreamWorks SKG's *The Road to Eldorado* and *The Prince of Egypt*. Linus Torvalds described Maya 3 as “the most complex and powerful 3-D graphics application ever to run on Linux.” In this article, I review installing and using Linux Maya 4, point out resources on the Web of interest to Maya animators and interview the team who ported Maya to Linux.

Enemy Stuka from Universal Pictures' *Captain Corelli's Mandolin*. Visible motion blur is a deliberate effect generated during rendering.

Maya 1 was first released in 1998 on SGI IRIX, where it is considered a killer application. In fact, Alias|Wavefront is a division of SGI. Maya for Windows NT2000 first appeared in 1999, and the first Linux version was Maya 3, released in March with support for Red Hat Linux. My installation won't be on the typical Red Hat but rather on Debian Woody with kernel 2.4.7. My hardware configuration is a homebrew Athlon 1.2GHZ, ASUS A7A266 motherboard, 256MB DDR, with 100GB of 7,200RPM IDE disk drives. For graphics we're running XFree86 4.1.0 with its accelerated open-source DRI driver on an ATI ALL-IN-WONDER RADEON. Building and installing 4.1.0 with the accelerated RADEON driver is a story of its own—to be covered next month.

From Steven Stahlberg's Animation Short

Improvements in Maya 4 include enhancements, optimizations and bugfixes. It has a new 3-D paint system. Changes were made to rendering, character animation, brush-and-paint tools, and games-related functionality. Enhancements to the Maya nonlinear, motion-editing technology include time warping, character merging, drag-and-drop, and character set editing. New character animation features include switching between forward kinematics and inversion kinematics, quaternion-based IK, motion trails, ghosting and a jiggle deformer to wobble character muscles. What that means in English is that it has every feature you can think of for an animation package and then some.



Maya 4 Opening View

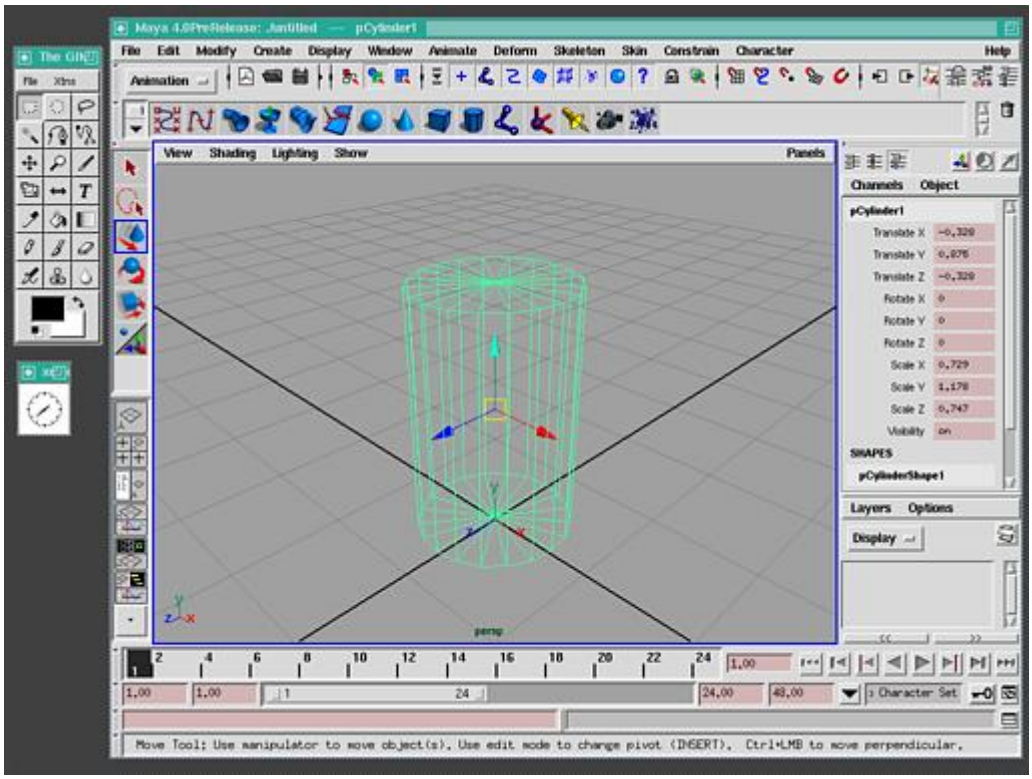
As commercial software for professional animators, Maya is priced accordingly. Maya Builder is $2,995 US, Maya Complete is $7,580 and Maya Unlimited is $16,000. A free 30-day evaluation period is available, and students are offered a half-price discount. Builder is configured primarily for game developers. It contains full-polygonal modeling, UV toolset (for working with textures) and basic animation features including path, scaling, keyframing, plus some of the deformation tools, some of the IK solvers (for realistic motion) and the full API. Builder provides no renderer because games have their own engine. By the way, most film animators use Pixar's RenderMan software for rendering their final output. Complete is Builder plus NURBS tools, all animation and IK tools, plus unlimited rendering. Unlimited is Complete plus Fur, Cloth and Live (camera extraction for match-moving mattes with live action). Before starting Maya you need to enter a separately provided serial number based on your machine's unique Ethernet MAC address into the license manager.

Linux Maya Unlimited is provided as a set of RPM files, about 143MB in the Maya 4 prerelease package I received. For Debian, we of course need deb files and not RPM. This provided an opportunity for me to try the package-converter Alien. Author Joey Hess says, “Alien converts between the rpm, deb, Stampede slp and Slackware tgz file formats. If you want to use a package from another distribution than the one you have installed on your system, you can use alien to convert it to your preferred package format and install it.” Alien is GPL and a package included with Debian.

I used the syntax **alien -k package_name.rpm** for Alien. When I ran Alien against Maya it created a deb but also generated a bunch of warnings of the form:

```
dpkg-shlibdeps: warning:
format of maya_lib_name.so is not recognized
```

where maya_lib_name.so was each of the shared libraries included with Maya. When I tried launching Maya it terminated, unable to find its shared libraries. Hess quickly set it right, “It's possible that the libraries are installed, but the necessary symlinks to let ldconfig find them are not installed.” He suggested manually running ldconfig, which did the trick. Besides running **ldconfig -v /usr/aw/maya4.0/lib**, I also added that directory name to my /etc/ld.so.conf file.



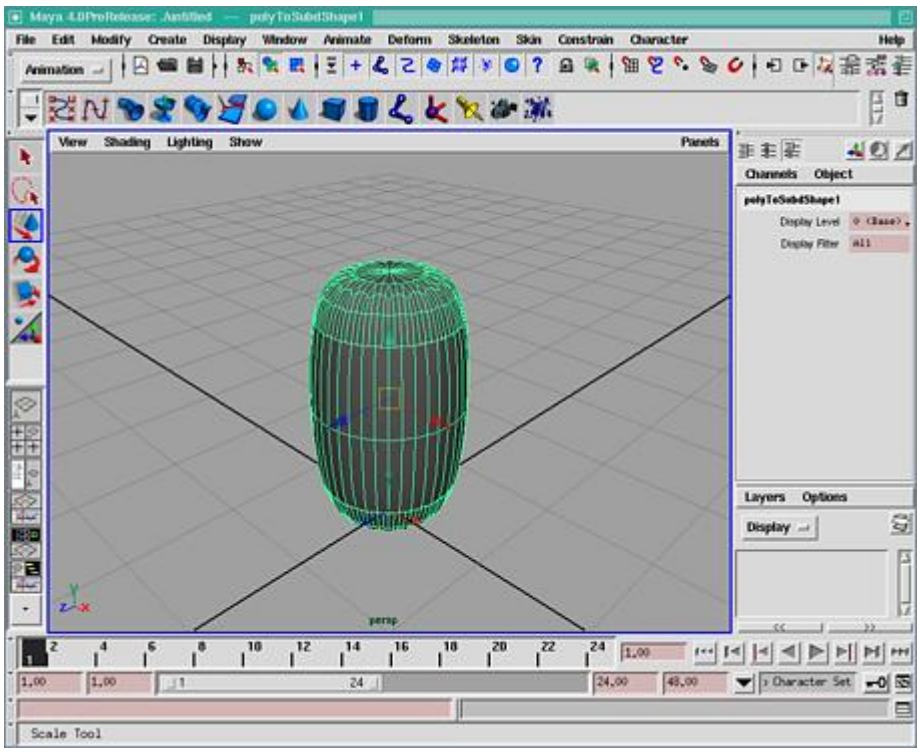
A Simple Polygon Cylinder

Maya is a large application and needs some time to load its plugins. Like the popular Linux image editor, the GIMP, it takes a while to load, but once it's up, it's fast. After Maya is loaded, you are presented with a grid plane in 3-D space. The functionality of Maya can be categorized roughly as modeling, animation, dynamics, shaders, rendering and plugins. Models are created from polygons, but the shapes may be faceted or smooth and organic. Working in a 3-D animation package is more like sculpting than drawing. You do need to be an artist.

The user interface for Maya centers on the Hotbox, a spider web of pop-up menus. Hotbox enables quick menu access by pressing the spacebar. You then drag from the center position to the menu desired. Although pull-down menus

are provided, many animators turn them off because using Hotbox is faster and more intuitive.

The Maya hotkeys are a refinement of the vi theory of key navigation: location, location, location. Hotkeys are placed according to speed of operation, like the QWERTY keyboard. The QWERTY key functions are Q for select, W for translate, E for rotate, R for scale, T for show manipulator and Y for last tool used. This system is very fast for trained operators but confusing at first. Note that everything in Maya is customizable. Using the built-in MEL scripting language you can change anything you don't like, even Maya's entire GUI. Maya's creators sometimes have to ask what application a Maya animator has up on his or her screen. It may not look anything like Maya.



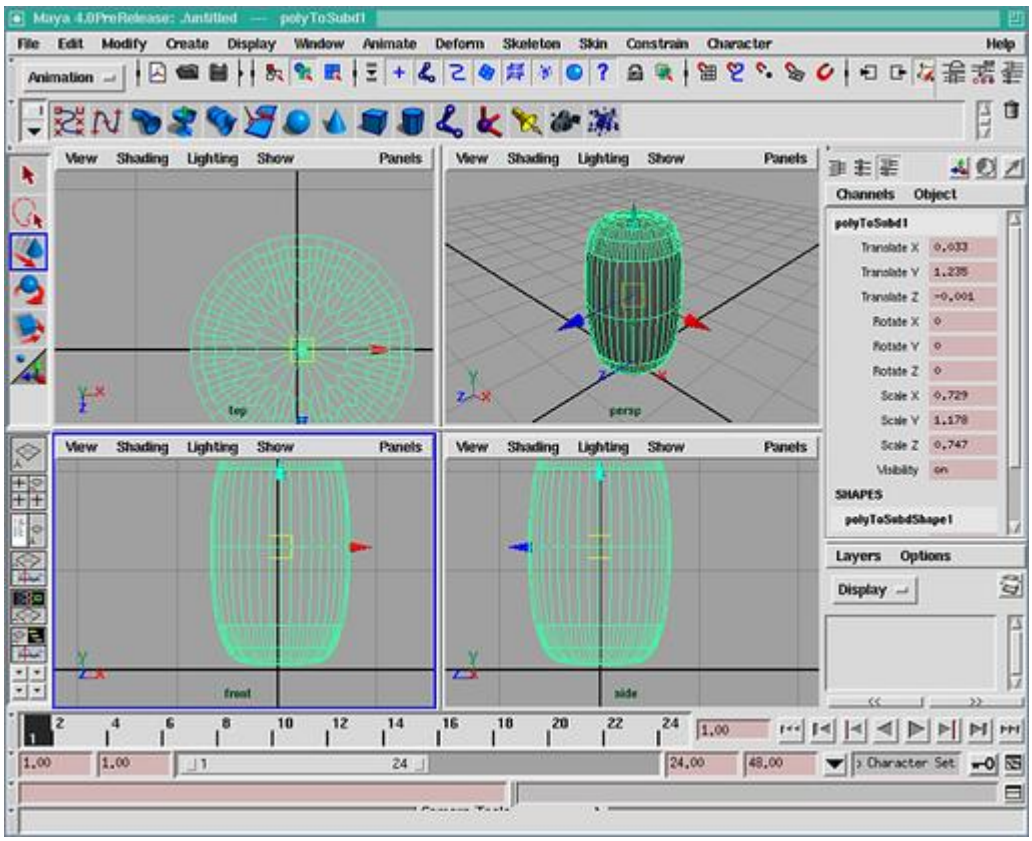
The Same Cylinder after Converting to SubD Keg Viewed in Four Perspectives Simultaneously

Let's make a simple figure. The icons across the top create basic shapes. Shapes may be polygons, subdivisions (SubD) or NURBS. Polygons are faceted shapes, while SubDs and NURBS are smooth. When shapes are created they pop to the origin at 0,0,0. Viewing the shape may require moving the view perspective to face the object. An easy way to do that is to use View® Frame Selection. The left row of icons contains the scale, move and rotate tools. Grabbing an object by one of its x-y-z handles manipulates it.

Looking from just one perspective tends to be misleading and causes mistakes. What happens is the side of an object you can see looks fine as you work on it, but the sides and back turn out to be way off. Quickly clicking the spacebar

switches to four-view mode. In this mode, you can see what you are doing from multiple perspectives, simultaneously.

An object created as a polygon can be converted into another type, such as SubD or NURBS. The advantage of a SubD surface is you can build a perfectly smooth complex object from a single primitive. When you animate a SubD it can't fall apart due to seams. NURBS may require stitching together separate pieces. Working with SubD surfaces is more like polygons in that areas may be extruded, for instance, to create fingers on a hand. You may also add levels of detail to a SubD to create complex features to local regions, such as the knuckles of a hand.



Clicking the spacebar switches to 4-view mode, vital for seeing your work from multiple perspectives simultaneously.

In working with Maya the initial reaction tends to be frustration. Although simple tasks are easy and intuitive, the sheer complexity of the tool defeats novices. The learning curve is steep. The software comes with excellent tutorials, but the books suffer from an assumption that you have seen Maya before. Following the tutorials (which are otherwise excellent) is a struggle in figuring out where an option box or menu referred to by the manual is located in the software. There are many menus and they drill many levels deep. There is even a search box to find a particular menu, a thoughtful but sobering feature.

For a detailed overview of the features of Maya and 3-D graphics in general see the book *The Art of Maya* published by Alias|Wavefront (2000). That may be your best bet if you decide to get just one book (besides the manuals), and it looks good on your coffee table besides. Third-party books include *Mastering Maya 3* (2001), *Digital Effects Animation Using Maya* (1999) and *Maya Illuminated: Games* (2001). There aren't drastic changes in Maya 4, so books on version 3 are still usable.

Searching for a Menu in Maya

The Web provides tons of information about Maya, starting with Alias|Wavefront's own site. Besides descriptions of Maya being used in movie productions, it lists events, job postings, industry links, listservs, user groups and HOWTO guides. You can download shaders, source images, props, MEL scripts, paint effects, sample projects, plugins, screensavers and desktop pictures there. Another site with a lot of content is HIGHEND3D. And, 3D CAFE says they are the world's largest web site for computer graphic artists. Both sites have large sections devoted to Maya. Magazines devoted to 3-D graphics include *Computer Graphics World*, *CGI*, *3D World* and *Computer Arts*, all of which have web sites.

There's an attractive gallery of projects done in Maya at users.pandora.be/gds/maya/gallery.htm. If you are interested in realism you may want to visit Steven Stahlberg's gallery and check out the tutorials on creating human faces at optidigit.com/stevens/howto.html.

General manager of Maya engineering Kevin Tureski says the reason for Maya on Linux is quite simple: “Demand for Linux became a dull roar. The Visual Effects Society, which represents our strategic customers, had a meeting with us about a year ago expressing a strong desire for Linux.” VES members are from the major motion picture companies.

According to Tureski, Maya is about 20 million lines of code, 15 thousand source files and 15 thousand classes. Linux Maya lead developer Wayne Arnold says they compiled using Red Hat 6.2 and gcc 2.91.66, although Red Hat 7.1 is supported. Linux tools used with Maya include vi, Emacs, gdb and ddd.

Support for accelerated graphics was a sore point with porting to Linux. “Only in March of this year has Linux become viable to fully support our configuration”, says Tureski. “Maya is a very large application. The OS and OpenGL are pushed to the limit.” Arnold says that drivers for HP fx10, ATI FireGL2 and NVIDIA are supported. The RADEON and NVIDIA open-source drivers are not. “We spent a lot of time with graphics card driver manufacturers getting Maya fully supported”, says Arnold. “The biggest problem was getting the correct visual 24-bit. There were a lot of memory leaks and other bugs to fix in drivers because they were so new. Some areas of pixel-type operations and triangles hadn't been hardware accelerated.” New code was added to Maya to remove the requirement for hardware overlay in order to provide compatibility even with (unsupported) nonaccelerated drivers.

Red Hat on HP was the initial configuration for Linux Maya, and other PC machines are described as being in various states of almost there. “Audio was a challenge and continues to be”, says Tureski. They are writing to the OSS sound API. Maya is almost equivalent on Linux to the Maya on Windows and IRIX. One thing missing is the ability to read AVI, QuickTime and MPEG files. On the subject of QuickTime players, ILM Director of Research Andy Hendrickson says that their Linux QuickTime code library may become open source.

DreamWorks SKG is using Maya on Linux for production of their next feature animation motion picture, *Spirit: Stallion of the Cimarron*, as described in this column last month. Many other studios are switching to Linux for their animation and special effects desktops. Thanks in part to Maya, Linux is becoming the technical desktop of choice for the film industry.

Resources

Robin Rowe (robin.rowe@movieeditor.com) is a partner in MovieEditor.com, a technology company that creates internet and broadcast video applications. He has written for *Dr. Dobb's Journal*, the *C++ Report*, the *C/C++ Users Journal* and *Data Based Advisor*.

Archive Index Issue Table of Contents

Advanced search

Modeling Seismic Wave Propagation on a 156GB PC Cluster

Dimitri Komatitsch

Jeroen Tromp

Issue #90, October 2001

California Institute of Technology builds a ground-shaking, 156-box, dual-processor cluster.

Large earthquakes in densely populated areas can be deadly and very damaging to the local economy. Recent earthquakes in El Salvador (magnitude 7.7 on January 13, 2001), India (magnitude 7.6 on January 26, 2001) and Seattle (magnitude 6.8 on February 28, 2001) illustrate the need to understand better the physics of earthquakes and motivate attempts to predict seismic risk and potential damage to buildings and infrastructures.

Strong ground shaking during an earthquake is governed by the seismic equations of motion, which support three types of waves: pressure (or sound), shear and surface waves. Numerical techniques can be used to solve the seismic wave equation for complex three-dimensional (3-D) models. Two major classes of problems are of interest in seismology: regional simulations (e.g., the propagation of waves in densely populated sedimentary basins prone to earthquakes, such as Los Angeles or Mexico City) and the propagation of seismic waves at the scale of the entire Earth. Every time an earthquake occurs, these waves are recorded at a few hundred seismic stations around the globe and provide useful information about its interior structure.

Numerical Technique

At the Seismological Laboratory at the California Institute of Technology, we developed a highly accurate numerical technique, called the Spectral-Element Method, for the simulation of 3-D seismic wave propagation. The method is based upon the classical finite element method widely used in engineering.

Each of the elements contains a few hundred points, solves the seismic wave equation on a local mesh and communicates the results of its computations to neighbors in the mesh. To model seismic wave propagation in the Earth, we create a mesh of the globe, which we divide into a large number of slices (see Figures 1 and 2). Each slice contains a large number of elements (typically several tens of thousands). The objective is to run the calculations on a parallel computer because the size of the mesh makes it impossible to run our application on a shared-memory machine or a workstation. Therefore, the method is perfectly suited for implementation on a cluster of PCs, such that each PC handles a subset of all the elements of the mesh. We use message-passing techniques to communicate the results between PCs across the network. This idea of parallel processing under Linux has developed rapidly in the scientific community (see the articles by M. Konchady and R. A. Sevenich listed in Resources).

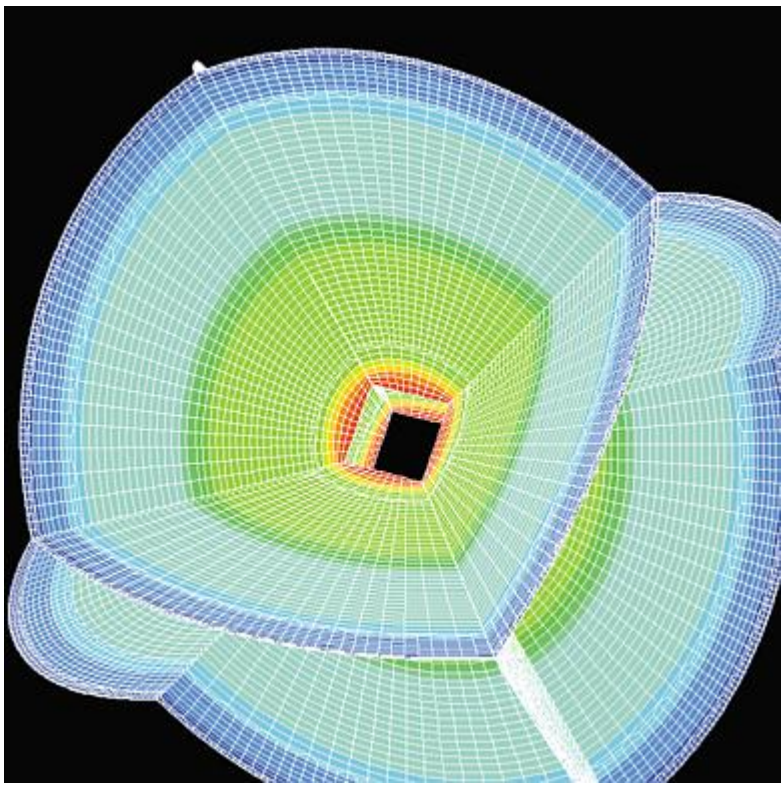


Figure 1. Mesh of the Globe

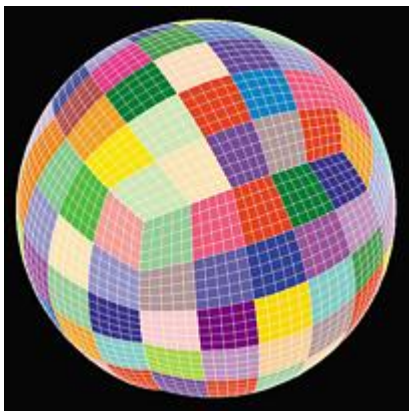


Figure 2. Slices Assigned to Processors

Research on how to use large PC clusters for scientific purposes started in 1994 with the Beowulf Project of NASA (beowulf.org), later followed by the Hyglac Project at Caltech and the Loki Project at Los Alamos (see Tom Sterling and collaborators' book *How to Build a Beowulf* and cacr.caltech.edu/resources/naegling). Hans-Peter Bunge from Princeton University was among the first to use such clusters to address geophysical problems, and Emmanuel Chaljub from the Institut de Physique du Globe in Paris, France introduced the idea of using message passing to study wave propagation in the Earth. Clusters are now being used in many fields in academia and industry. An application to a completely different field, remote sensing, was presented in a recent issue of the *Linux Journal* by M. Lucas (see Resources).

Hardware

For our project we decided to build a cluster from scratch using standard PC parts. The acronym COTS, for commodity off-the-shelf technology, is often used to describe this approach. The main constraint was that we needed a large number of PCs and a lot of memory because of the size of the meshes we wanted to use in our simulations. Communications and I/O are not a big issue for us since the PCs spend most of their time doing computations, and the amount of information exchanged between PCs is always comparatively small. Therefore, our particular application would not benefit significantly from the use of a high-performance network, such as Gigabit Ethernet or Myrinet. Instead, we used standard 100Mbps Fast Ethernet. Due to the large number of processors required (312 in total), we used dual-processor motherboards to reduce the number of boxes to 156, thus minimizing the space needed for storage (and the footprint of the cluster). This structure impacts performance because two processors share the memory bus (which causes bus contention but reduces the hardware cost) since only one case, motherboard, hard drive, etc., are needed for two processors. We ruled out the option of rackmounting the nodes, essentially to reduce cost, but chose to use standard mid-tower cases on shelves, as illustrated in Figure 3. This approach is sometimes given the name LOBOS (“lots of boxes on shelves”). The shelving system was placed in a computer room already equipped with a powerful air-conditioning system and 156 dual-processor PCs.

Figure 3. 156 Dual-Processor PC Cluster. The boxes are connected using a standard 100Mbps Fast Ethernet network (the green and blue cables). In the back, one can see the 192-port Cisco switch. The height of the shelving system is approximately eight feet.

Deciding between Pentium IIIs and AMD Athlon processors was difficult. The Athlon is said to be faster for floating-point operations, which is the main type of operation used in most scientific applications, including ours. At build time, no dual-processor Athlon motherboard was available. As mentioned above, using single nodes would have increased the total cost of the cluster. For this reason, we selected the Pentium III.

It is tempting to use the latest technology when assembling a PC. However, new processors are more expensive than six-month-old technology and offer a small increase in performance. Three- to six- month-old processors provide the best trade-off between price and performance. We used 733MHz processors when we assembled the machine in the summer of 2000.

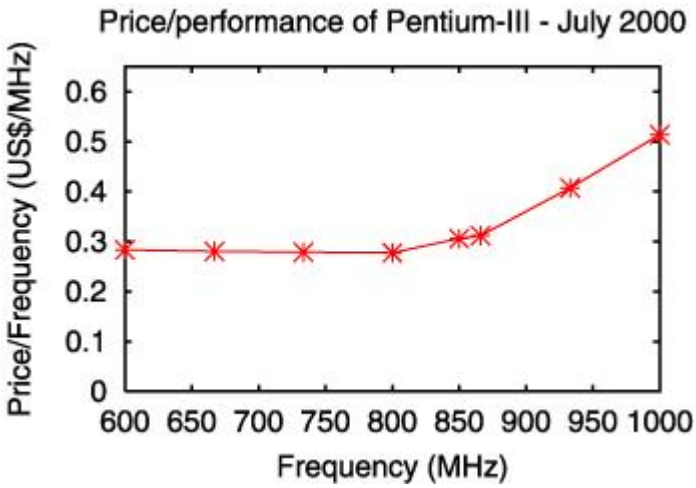


Figure 4. Price/Performance Ratio for the Pentium III

Figure 4 shows the ratio between price and performance for the Pentium III processor. The prices shown are an average of typical prices from retailers in the US. As one can see, old processors are cheap but relatively slow. New

processors are faster but much more expensive. The optimal price/ performance ratio is obtained in between.

We decided to put the maximum possible amount of memory on the motherboards, i.e., fully populate the memory slots with 1GB of RAM per PC for a total of 156GB of memory in the cluster. Each PC is also referred to as a “node” or “compute node”. Note that memory represents more than 50% of the total cost of the cluster.

The rest of the hardware is fairly standard: each PC has a Fast IDE 20GB hard drive, a Fast Ethernet network card and a cheap 1MB PCI video card, which is required for the PC to boot properly and can be used to monitor the node if needed. We use high-quality, mid-tower cases with ball-bearing case fans because the mechanical parts in a cluster, such as fans and power supplies, are the most likely to fail. Note that the total disk space in the cluster is enormous (20GB × 156 = 3,120GB = 3TB). To further reduce the cost of the cluster and to have full control over the quality of the installed parts, we decided to order the parts from different vendors and assemble the nodes ourselves, rather than ordering pre-assembled boxes. It took three people about a week to assemble the entire structure. One PC, called the front end, has a special role in the cluster: it contains the home filesystems of the users (SCSI drives NFS-mounted on the other nodes with the autofs automounter), the compilers, the message-passing libraries and so on. Simulations are started and monitored from this machine. The front end is also used to log in to the nodes for maintenance purposes. The nodes are all connected using a 192-port Catalyst 4006 switch from Cisco, which has a backplane bandwidth of 24Gbps (see Figure 5).

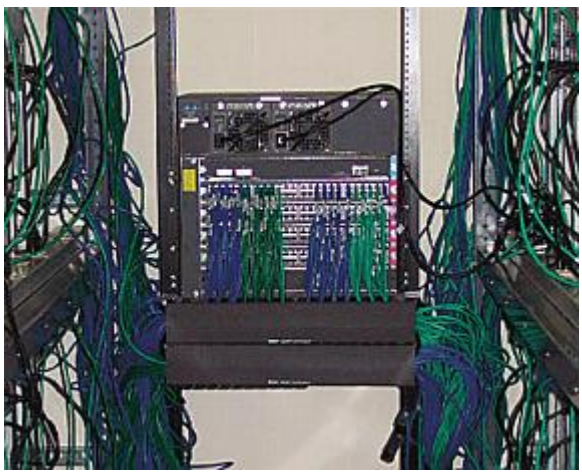


Figure 5. Catalyst 4006 Cisco Switch

Software Configuration and Code Development

All the nodes in the cluster run Linux Red Hat 6.2. Linux corresponds perfectly to the demands of our application; we require very high reliability because the machine is being used by researchers who need their jobs to run without

having to worry about nodes crashing. We have not had a single system crash since the machine was built nine months ago. The operating system needs to be tuned to the hardware in order to reach maximum performance; with the open-source philosophy, we have been able to recompile the kernel with a minimal set of options corresponding to our hardware configuration. We recently installed the 2.4.1 kernel, which has much better support for dual-node SMP machines than the 2.2 kernel. The performance is excellent; by switching from 2.2 to 2.4, the CPU time of our application has decreased by 25%. In terms of network configuration, the 156 nodes are on a private network of 192.168.1.X addresses. For security reasons, the cluster is not connected to the outside world, and all the post-processing and analysis of the results is done locally on the front end. We use rdate once a day in the cron table of each node to synchronize the time with the front end.

The biggest price we had to pay for the use of a PC cluster was the conversion of an existing serial code to a parallel code based on the message-passing philosophy. In our case the price was substantial because our group is composed of researchers who are not professional programmers. This situation meant we had to dedicate a few months to modifying several tens of thousands of lines of serial code. The main difficulty with the message-passing philosophy is that one needs to ensure that a control node (or master node) is distributing the workload evenly between all the other nodes (the compute nodes). Because all the nodes have to synchronize at each time step, each PC should finish its calculations in about the same amount of time. If the load is uneven (or if the load balancing is poor), the PCs are going to synchronize on the slowest node, leading to a worst-case scenario. Another obstacle is the possibility of communication patterns that can deadlock. A typical example is if PC A is waiting to receive information from PC B, while B is also waiting to receive information from A. To avoid deadlocking, one needs to use a master/slave programming methodology.

We use the MPI (message-passing interface) library to implement the message passing. Specifically, we installed the open-source MPICH implementation developed at Argonne National Laboratory (see the 1996 article by W. Gropp and collaborators, available at www-unix.mcs.anl.gov/mpi/mpich). This package has proven to be extremely reliable with Linux. MPI is becoming a standard in the parallel-computing community. Many features of MPI are similar to the older PVM (parallel virtual machine) library described in the article of R. A. Sevenich in *Linux Journal*, January 1998.

An additional difficulty with our project was the amount of legacy code we had to deal with. A lot of modern codes are based on libraries that contain legacy code developed in the 1970s and 1980s. Almost all scientific libraries were written in Fortran77, the language of choice at that time; use of C was not yet

widespread. We decided not to convert the 40,000+ lines of code to C, rather we upgraded from Fortran77 to the modern Fortran90. The new version has dynamic-memory allocation, pointers, etc., and is back-compatible with Fortran77. We wrote a Perl script to perform most of the conversion automatically, fixing a few details by hand and changing memory allocations from static to dynamic. Unfortunately, to our knowledge no free Fortran90 compiler is currently available under Linux. The GNU g77 and f2c packages only support Fortran77. So, we had to buy a commercial package, pgf90 from The Portland Group, pgroup.com. This is the only non-open-source component in our cluster.

A limitation of PC clusters is the problem of system administration and maintenance. Using hundreds of PCs, one increases the probability of hardware or software failure of nodes. In the case of a hardware problem, the nice thing about PCs is that parts are standard and can be bought and replaced in a matter of hours. Therefore, the cost associated with maintenance is low compared to expensive maintenance contracts researchers used to need for classic supercomputers.

Software maintenance is more of an issue—with 156 nodes, how do you make sure they are all working properly? How do you install new software? How do you monitor the performance of a job that is running? When we installed the cluster, we wrote scripts that collected information from the nodes by sending rsh commands. Since then, universities like Berkeley and companies like VA Linux have developed efficient software packages for cluster monitoring and have made them open source. We use a node-cloning package called SystemImager from VA Linux (valinux.com) to do software upgrades. With this package we only need to upgrade the first node manually. Then the package uses rsync and tftp commands to copy (or clone) the changes to the other 155 nodes in a matter of minutes. To monitor the cluster and the jobs that are running, we use the Ganglia package from Matt Massie at Berkeley (millennium.berkeley.edu/ganglia), which is a fast and convenient package that uses a system of dæmons to send information about the state of each node to the front end, where it is gathered and displayed.

In Figure 6, we show a Tcl/Tk interface to the Ganglia package. The GUI we use is based on another open-source package, bWatch by Jacek Radajewski (sci.usq.edu.au/staff/jacek/bWatch). We modified it for our needs and to use Ganglia instead of standard rsh commands for much faster access to the nodes. Also, VA Linux has recently released the VACM package (VA Cluster Management), which we have not yet installed.

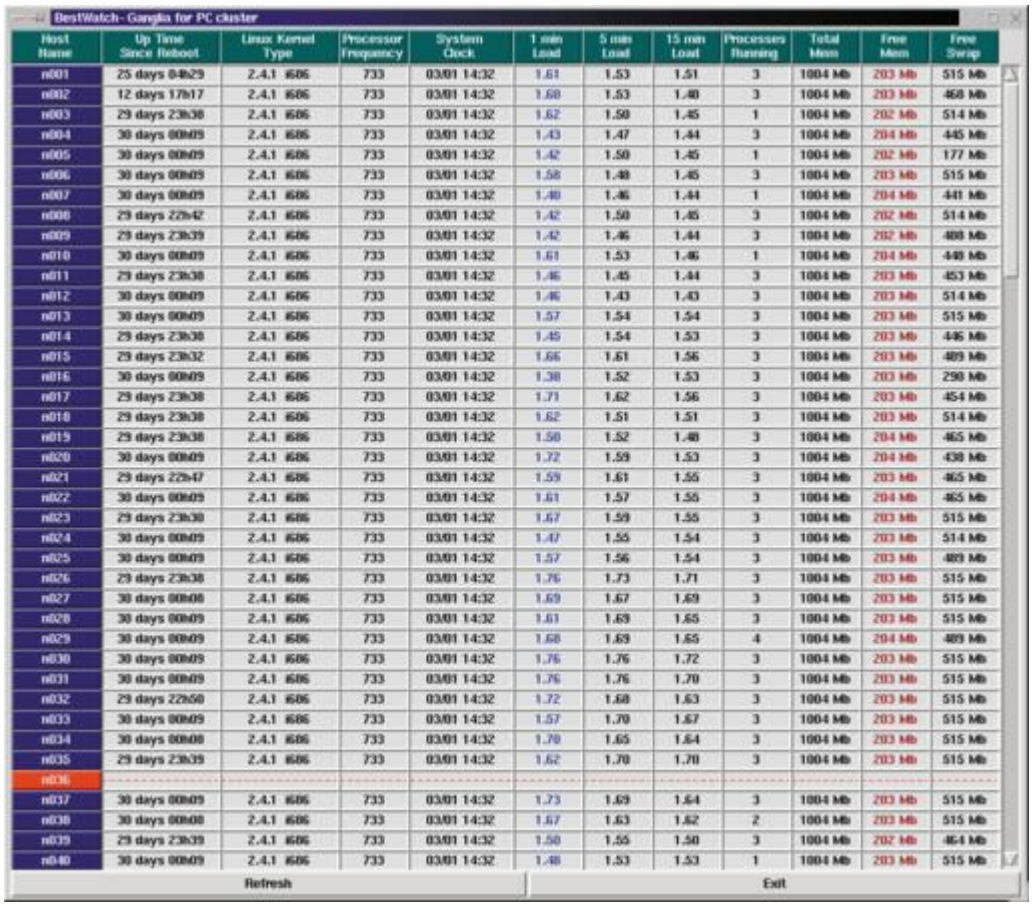


Figure 6. Monitoring with Ganglia

Bolivia Shakes and Moves

On June 9, 1994, a huge earthquake with a magnitude of 8.2 on the open Richter scale occurred in Bolivia, at a depth of 641 km (400 miles). Most earthquakes occur at much shallower depths, usually less than 30 kilometers. This event in Bolivia was one of the largest deep earthquakes ever recorded. Due to its unusual characteristics, this earthquake has become the subject of numerous studies in the seismological community. We tried to simulate this event on our cluster.

Figure 7 shows a still of the ground shaking (displacement of the Earth at a given location due to the passage of a seismic wave generated by the earthquake). The epicenter in Bolivia is indicated by the purple triangle. The waves travel inside and along the surface of the Earth. They can be seen propagating across the United States, for instance. A permanent displacement is visible at the surface of the Earth around Bolivia, extending as far as the Amazon river to the north. This effect, which was recorded by several seismic stations in Bolivia, is called the “static offset”. The earthquake was so big that it moved the ground permanently by a few millimeters. The vertical displacement reached up to 7mm, i.e., ¼“ to the south). It is correctly reproduced by our code.

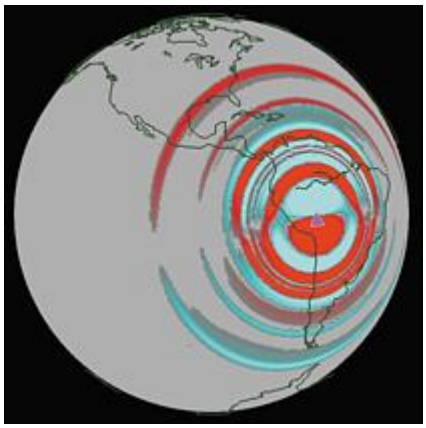


Figure 7. Seismic Waves during the 8.2 Bolivia Earthquake

Due to the fact that the waves travel all around the globe, seismic recording stations in other countries were able to detect the Bolivia earthquake. Figure 8 shows an actual record from a station in Pasadena, California and the corresponding record simulated by our method. Again, the agreement is satisfactory. At each time step, this simulation required solving a system of equations with 500 million unknowns (also called the degrees of freedom of the system). Simulating the propagation of seismic waves for an hour and a half after the earthquake took 48 hours on the cluster using half of the nodes (150 processors).

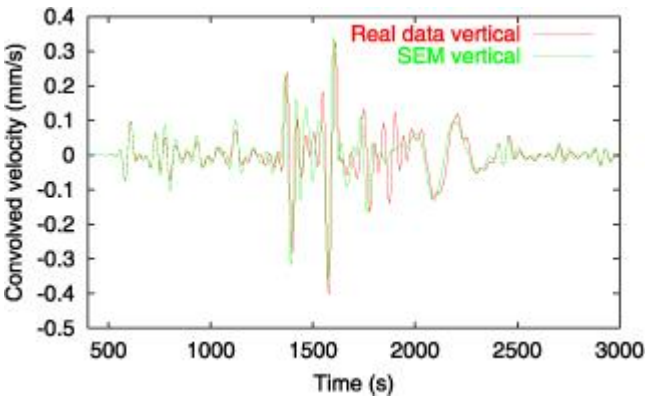


Figure 8. Vertical Velocity as Recorded in Pasadena

Needless to say, our research has benefited tremendously from the power and the reliability of Linux and from the open-source philosophy. Using a large cluster of PCs, we are able to simulate the propagation of seismic waves resulting from large earthquakes and reach unprecedented resolution.

Acknowledgements

Luis Rivera provided invaluable information and help for this project. We thank Jan Lindheim, Tom Sterling, Chip Coldwell, Ken Ou, Jay Nickpour and Genevieve Moguilny for discussions regarding the structure of the cluster. Matt Massie added several options to his nice Ganglia package to help us monitor more

parameters on our cluster. Rusty Lusk provided some useful insight about running MPICH on large clusters.

Resources

Dr. Dimitri Komatitsch is a senior researcher in the Division of Geological and Planetary Sciences at the California Institute of Technology. His interests are applied mathematics, numerical analysis and the application of computer science to problems in geophysics and seismology.

Dr. Jeroen Tromp is a professor in the Division of Geological and Planetary Sciences at the California Institute of Technology. He is interested in theoretical seismology, in particular seismology at the scale of the Earth. Recently he has focused on numerical modeling of seismic wave propagation.

Archive Index Issue Table of Contents

Advanced search

Distribution Upgrades

David A. Bandel

Issue #90, October 2001

David enjoys having it all his way—free meds, burning CDs and more.

As I write this it looks like all the players have weighed in with their new distribution offerings. I've checked out my favorites and found them all wanting, but still a site better than those from the past. This time, though, I've noticed what I would consider a lot more griping and general discontent than before among the user community. I always figured new releases were a time of rejoicing that someone else went to the trouble of upgrading everything for me. Last time around I seem to recall a lot of whining about the distros installing with everything in the world turned on (Telnet, FTP, etc.). The distros heard this and tightened down. Now I hear the same gripe, only this time it's because nothing (or almost nothing) is turned on by default. Guess some things just never change. But I for one am enjoying the moment, discovering where the engineers squirreled away all the scripts and configuration files this time. It'll soon be old hat. And when I've got the half-dozen distros I'm running here down pat, I guess I'll just have to go check out the other 112-plus. I feel like a kid at Burger King, having it all my way. Gee, it's great to be running Linux.

FreeMED freemed.org

The FreeMED program is striving for ISO9000 compliance with a complete package for a medical practice or hospital. It's not there yet, lacking several major features, but looks good, performs well and is quite easy to install. For security, just drop this into your secure web document root and go. Requires: web server w/ PHP4 and MySQL support, MySQL server, phpwebtools, web browser.

EthStatus ethstatus.calle69.net

If you want to monitor the status of one or more Ethernet interfaces in a system, this small utility is for you. It will show you, both graphically and in

numbers, what your interface is doing and even whether it's up or down. The author has included a small utility to set up startup at boot. But even if you don't have one of the distributions mentioned, you can set it up. Output can be sent to an unused VT for monitoring at any time by anyone. Requires: libncurses, glibc.

Sysmon sysmon.org

If you have a lot of systems you need to watch, Sysmon can handle them all fairly easily. Designed for a NOC (network operations center), you set up one system as your monitoring station running the sysmon dæmon, and you can check it via the sysmon client from anywhere on your network. The most difficult part is the configuration, which isn't well documented, so novices may find setup daunting. Requires: libresolv, libnsl, libncurses, glibc.

wavemon jm-music.de/projects.html

If any of you use a wireless network, you are almost certainly aware of the paucity of tools for these cards. Well, for those using the WaveLAN or similar, at least one tool is available. While wavemon isn't a panacea; it's a start. It works for a myriad of cards, including the Lucent Orinoco and many others. Now, if some tool would just allow this card to be used under Linux as an access point. Requires: libm, libncurses, glibc.

popaccess ftp.innercite.com/pub/popaccess

I know many administrators face the problem of allowing only certain authorized individuals access to their mail server without making it a relay for low-life spammers. But when your road warriors are traveling, or, like me, you have folks who access the server from all over the world, you need something. What I liked about popaccess is that it's all in one file, not three or four. Configuration is simple and it works. Requires: Perl, perl module IO::Seekable.

SimpleCDR ogre.rocky-road.net/cdr.shtml

I have nothing against xcdroast. I also have nothing against X. But sometimes working from a TTY is faster or easier, so often I do. SimpleCDR works in most any environment but is particularly well suited to a VT. It won't burn your CD any faster, but you don't need to fire up X to use it. Requires: libncurses, libstdc++, libm, glibc.

multiCD multicd.rmdashrf.org

Sometimes, you just need to create multiple CDs. Most systems I've worked on would require several CDs if you wanted to use them for backups. **multiCD** was

made just for that purpose. Even as it's burning one CD, it can be creating the next so that burning can continue uninterrupted. For obvious reasons, this won't work on a slow system. Requires: Perl.

IC-RADIUS radius.innercite.com

If you require a RADIUS server, for whatever purpose, you'll at least want to take a look at IC-RADIUS. Once installed, all administration is performed through a web browser (though you'll want a secure server to perform remote administration). You can create groups, users, view statistics and more. Network logins via dialup are very fast. Requires: MySQL, libmysqlclient, libm, libcrypt, libnsl, libz, glibc.

Until next month.

David A. Bandel (dbandel@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is coauthor of Que Special Edition: Using Caldera OpenLinux.

Archive Index Issue Table of Contents

Advanced search

Naming Open-Source Software

Lawrence Rosen

Issue #90, October 2001

The value and importance of trademarks.

Suppose your favorite lunch spot offered you a glass of sweetened, carbonated water, caramel-colored, with natural and artificial flavors and caffeine. Would you drink it? Would you want to know who made it? Would you want to know in advance that your drink will taste like the sweetened, carbonated water that you enjoyed so much several days earlier?

Now, supposed your favorite lunch spot offered you a Pepsi. Would your questions about quality (good or bad) and product consistency suddenly go away? What would you select if given the choice between a Coca-Cola and a Melvyn's Cola? Would you be more likely to select a product simply because you are familiar with its name and reputation?

We are all caught up in marketing. Television and other advertising media continually inundate us with product names that we almost automatically add to our vocabulary, as if they were words in the English language. But these product names are not words in the traditional sense, nor are they intended to be so used. Terms like Windows, Apple, Java, Apache, Linux, Jabber and JBoss are marketing names—trademarks. When carefully selected and protected, and through effective marketing, trademarks can become badges of quality, expertise and reputation.

Trademarks are potentially among the most valuable assets of companies that market free and open-source software. For some projects, a trademark is much more important than the license under which the software is distributed.

Trademarks are a form of intellectual property, but they are very different from patents, which protect invention; copyrights, which protect expression; and trade secrets, which protect a company's confidential information. Trademarks

are brand names of products or services. They serve to protect the reputation of a product.

More specifically, a trademark is a word, name, symbol or device used to identify a company's products and to distinguish those products from the competition. A service mark is the same as a trademark except it is used to identify and distinguish services rather than products. A trade name identifies a company rather than that company's individual products or services. The same symbol might function as a trademark, a service mark and a trade name, depending on the context in which it is used. Here I will generally use the term trademark and ignore the subtle distinctions of service marks and trade names.

A company doesn't acquire trademark rights simply by choosing a trademark or even by stating its intent to use it. Trademarks (or service marks) in the United States are established by actual use in conjunction with specific goods (or services). There are advantages to registering trademarks with the US Patent and Trademark Office, but registration alone doesn't make a trademark valid or valuable. Only actual use of a trademark on goods, and a marketing campaign to breed familiarity among consumers, can create a valid and valuable trademark.

Free and open-source products pose difficult marketing problems. The licenses under which such products are distributed require the distribution of source code and permit the creation and distribution of derivative works. It is difficult for a distributor of such products to compete on price alone because almost any knowledgeable company can undercut the price by simply copying the original software.

Trademarks can be particularly useful in this kind of environment. First, a company needs to demonstrate that its software is of high quality, reliable, efficient, feature-rich and user-friendly. It can promise continual enhancements, product support, user groups and undertake other goodwill-creating activities. Then over time, through marketing efforts, that company's customers will begin to associate its trademarks and service marks with those products and services. New or repeat customers will pay for goods and services they perceive to be worth the price, and they will select preferentially products whose trademarks they identify.

Trademarks can also limit certain kinds of forking of free and open-source code. While your software license may allow others to make derivative works of your code, the license doesn't allow (indeed, there is a legal reason why it cannot allow) others to apply your trademark to their derivative works.

In any industry where it is not technically difficult to create a new product, whether it be the cola beverage industry or the free and open-source software industry, trademarks and service marks can provide the best protection for a company's products and services.

We see this every day when customers order Coca-Cola and Pepsi rather than Melvyn's Cola. That is why customers buy Red Hat Linux rather than El Cheapo Linux. That is why Apache servers dominate the Web, rather than servers based on generic copies of the Apache code.

And that is why, if you are wise in selecting, perfecting and using your trademarks and service marks, you can protect the market for your free and open-source software products from competitors who merely want to copy your work and make money off your well-earned goodwill.

In future articles, I will describe how you should select a good trademark and explain what you need to do to use it and protect it properly.

Legal advice must be provided in the course of an attorney-client relationship specifically with reference to all the facts of a particular situation and to the law in your jurisdiction. Even though an attorney wrote this article, the information in this article must not be relied upon as a substitute for obtaining specific legal advice from a licensed attorney.

Lawrence Rosen is an attorney in private practice in Redwood City, California (rosenlaw.com). He is also executive director and general counsel for Open Source Initiative, which manages and promotes the Open Source Definition (www.opensource.org).

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

The Bazaar Way to Bet

Doc Searls

Issue #90, October 2001

Linux—more popular than Jesus? According to Google's authoritative search engine—yes.

If Google isn't already the number one search engine, it might as well be. What else is there? Even Yahoo gave up and started OEMing Google a few months back. They've got 10,000-plus Linux boxes sending out bots that crawl over 1.3 billion pages and bring back results so complete that every page is cached in searchable form. They're also on top of images and Usenet groups in a huge way, even though they hardly publicize either while they continue (at this writing) to tweak the systems publicly as beta.

Advertising on Google has more in common with your newspaper's classifieds than with anything on billboards, magazines or TV screens. This observation is meant as flattery. They are about the only form of advertising that enjoys positive demand, which means Google stands a chance of becoming the first outfit to create a form of web-based advertising that users welcome—no mean feat if they pull it off.

Their search results are relevanced (how's that for a neoverb?) by inbound links rather than by clever metatags and other sneaky HTML hacks on the searched pages. If anything, Google measures authority more than anything else. Since linking isn't an automatic matter with most text processing software, it takes a conscious effort for an author to point. Therefore, Google's thinking (roughly) goes, every link to a page is a vote by an interested human being for the value of the pointed-to page as a source.

Google calls its system “a unique combination of hardware and software”. Specifically, it's ten thousand boxes running Linux. We all know the technical answer to the “Why Linux?” question. There's an obvious economic one too: it's cheap to deploy on commodity hardware. More importantly, it lowers the threshold of deployment. It was easier for Google's creators to imagine the

service running on Linux than on anything else. But finally, Google provides a democratic answer with its own search results.

See, in addition to ranking search results, Google also lists the number of unique pages found to contain the search term (or terms). When I looked a few minutes ago (when putting together the *LJ* Index, where these sort of numbers are also posted), Google found 31.6 million pages containing the word “linux”. Let's put this in perspective:

- active x: 2,350,000
- python: 2,080,000
- gates: 3,020,000
- kde: 3,560,000
- gnome: 3,720,000
- perl: 7,650,000
- jesus: 8,800,000
- boy: 10,800,000
- solution: 13,300,000
- market:31,200,000
- girl: 13,600,000
- microsoft: 20,200,000
- god: 24,300,000
- sun: 25,500,000
- sex: 28,400,000
- linux: 31,600,000
- business: 86,900,000
- have: 231,000,000
- naked:8,080,000
- Tcl:1,910,000
- foobar: 113,000
- hampster dance:8,930

On that last one, the top two (out of 231 million) were LinuxGames.com and Richard Stallman's Why Software Should Not Have Owners page.

If markets are conversations, what does this say about the Linux market? Is this market—our bazaar—really bigger than sex and Microsoft?

Apparently.

Why? Because there's a lot to talk about and keep talking about; because it constantly changes. No matter how important Microsoft continues to make itself (by expertly leveraging its incumbent importance), the company's protectiveness about its intellectual property puts a lid on the quantity of Microsoft subjects one can talk about.

There is a limit to how deeply one can enjoy expertise in—or involvement with—Microsoft software. Where a significant hunk of Microsoft intellectual property is nobody's business, all of Linux is anybody's business.

Which means Linux, as a topic of expertise, is better for business than Microsoft, or any other commercial software company that limits conversation around the very software on which it wants everybody to depend.

This is what we need to remember when we speculate about the viability of Linux companies. A lot of them got clobbered in the dot-com bust, along with everybody else. If we want to put our trust in market forces, it's pretty hard to ignore one of the world's biggest conversations.

Doc Searls is senior editor of *Linux Journal* and coauthor of *The Cluetrain Manifesto*. His next book will be *Real Markets: What They Are and How They Work*.

Archive Index Issue Table of Contents

Advanced search

The Robots Are Coming, The Robots Are Coming

Rick Lehrbaum

Issue #90, October 2001

Linux takes over the world, one step at a time—with smaller footprints and larger funding.

Isamu stands 53 inches tall, weighs 121 pounds and walks at over one mile per hour. Not only that, Isamu climbs up and down stairs, carries four-pound objects in its handlike grippers, and even recognizes human faces via its dual-camera stereo vision system. Isamu also has a brain, which consists of a dual-Pentium embedded computer running RTLinux.

Isamu is the product of a joint project of the University of Tokyo's Jouhou System Kougaku Laboratory (JSK Lab) and the Aircraft and Mechanical Systems Division of Kawada Industries, Inc. (Tokyo, Japan). The goal of the project is to develop testbeds for research on “human interactive motion control technology”. For example, Kawada Industries plans to explore possible

commercial applications in markets such as construction systems, disaster relief, aids for the disabled, rehabilitation and training devices, and amusement.

In order to mimic humanlike movement, Isamu has thirty-five degrees of freedom: six for each leg, one for each foot (a toe joint), seven for each arm, one for each gripper, two for the neck and three for the eyes. The on-board computer, equipped with dual 750MHz Pentium III processors running RTLinux, provides real-time servo and balance compensation, and coordinates the robot's 3-D vision and motion-planning software modules.

Thanks to an ample battery pack, a wireless Ethernet interface and the powerful on-board computer, Isamu can operate without the need for external cables or constant human intervention. A joystick can be used to direct the robot's movements when human control is desired.

Isamu's bipedal walk, control-system software was developed by the JSK Lab, while the hardware and robotics structures, including the servo-based, level control system, were developed by Kawada Industries. Kawada applied aircraft technologies to the body frame, resulting in a strong and light structure. Visit the JSK Lab's web site (jsk.t.u-tokyo.ac.jp), where you can view some amazing videos showing some of the things Isamu can do.

Linux Upgrade for the Palm III

Empower Technologies has announced what it calls “the world's first major operating system upgrade for Palm IIIx and IIIxe handhelds”. A preliminary demo version of Linux DA is available for free download from the company's web site. Although the demo version is limited to only the most basic PDA functions (address, schedule, calculator, notepad and a few games), the commercial version will add functions such as web browsing, e-mail, multimedia and a lot more. The company says its Linux DA was derived from the standard Linux kernel, with the addition of homegrown software for a GUI, database, power management, boot loader, Flash loader, data sync, IrDA and PIM functions. See empower-technologies.com for details.

Transvirtual Nets Four Million Dollars in Investment, Launches

XOE Device Platform

Transvirtual Technologies, Inc. has closed a four-million-dollar funding round and is using much of the new capital to launch a new Java/XML-based information appliance platform, called eXtensible Operating Environment (XOE). XOE (pronounced “zo' ee”) evolved Transvirtual's PocketLinux Linux/Java environment for handheld computers, along with a number of XML-based

client and server functions. The result, according to Transvirtual, is “a faster, cheaper and more flexible solution specifically engineered for small, resource-constrained information appliances such as PDAs, web-enabled mobile phones, automotive telematics and TV settop boxes”. Although initially targeted to embedded Linux-based devices, XOE will eventually support other embedded OSes as well. See transvirtual.com for details.

Linux in 27 × 27mm

Axis Communications, maker of the ETRAX system-on-chip, has integrated over 50 components into a multichip module (MCM) in a standard 27 × 27mm PBGA IC package. Axis says just two external components are needed to end up with a fully functional Linux system: a 20MHz crystal and a source of 3.3 V power. The new MCM contains Axis' ETRAX 100LX RISC-based system-on-chip processor, plus all the most common components normally put in designs based on the device, such as DRAM, Flash, an Ethernet transceiver and “glue” components. Axis says it has run Linux and a small application program successfully, entirely within the MCM's built-in 2MB Flash and 8MB SDRAM. A development board and sample schematics of typical applications for the device are available. Samples are expected in this month. Pricing has yet to be finalized but is expected to be in the range of $50 US (10K units) to $75 (single units). See developer.axis.com for details.

email: rick@linuxdevices.com

Rick Lehrbaum (rick@linuxdevices.com) created the LinuxDevices.com “embedded Linux portal”. Rick has worked in the field of embedded systems since 1979. He cofounded Ampro Computers, founded the PC/104 Consortium and was instrumental in creating and launching the Embedded Linux Consortium.

Archive Index Issue Table of Contents

Advanced search

Microlite BackupEDGE Version 01.01.08

Charles Curley

Issue #90, October 2001

BackupEDGE from Microlite is an excellent backup product.

BackupEDGE from Microlite is an excellent backup product. It has a flexible command-line and menu interface, will back up over networks and can be used for disaster recovery. Documentation is extensive and readily available. It is a proprietary product, priced competitively. BackupEDGE uses existing utilities in Linux, such as rsh or ssh and crontab for its operations, making it easier to adjust BackupEDGE to live with other programs.

Installation

The product is delivered on CD-ROM. If your target computer does not have a CD-ROM drive, you can make installation floppies on another computer from the CD-ROM. You can even make floppies using that other operating system from Redmond, and you can get a 60-day evaluation copy via FTP. If you decide to buy, Microlite will send you a key that makes your evaluation copy permanent.

The installation script is easy to use. It is a command-line script with character graphics (see Figure 1), similar to the main program. Arrow keys move the cursor from field to field, but the tab key may not. I found the inconsistent responses to the Tab key to be disconcerting. For text entry, you can toggle between insert and overwrite modes using the Insert key on a standard PC keyboard, which is very nice. I used that capability to prepend the type of drive to the description of each tape drive. On-line help is readily available during both installation and normal usage (see Figure 2).

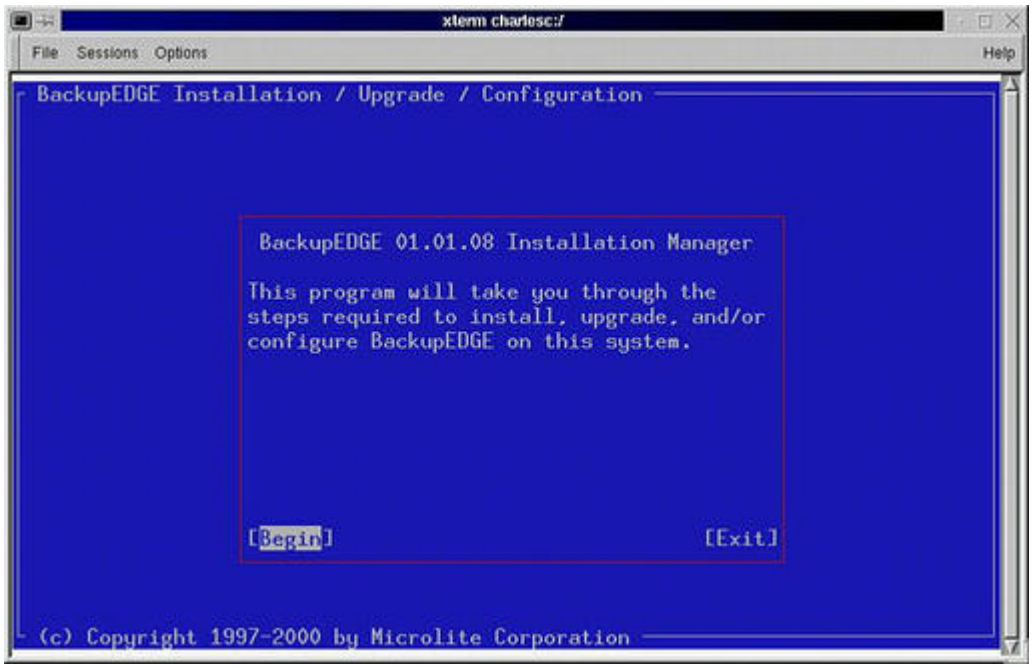


Figure 1. Installation Script Welcome Screen

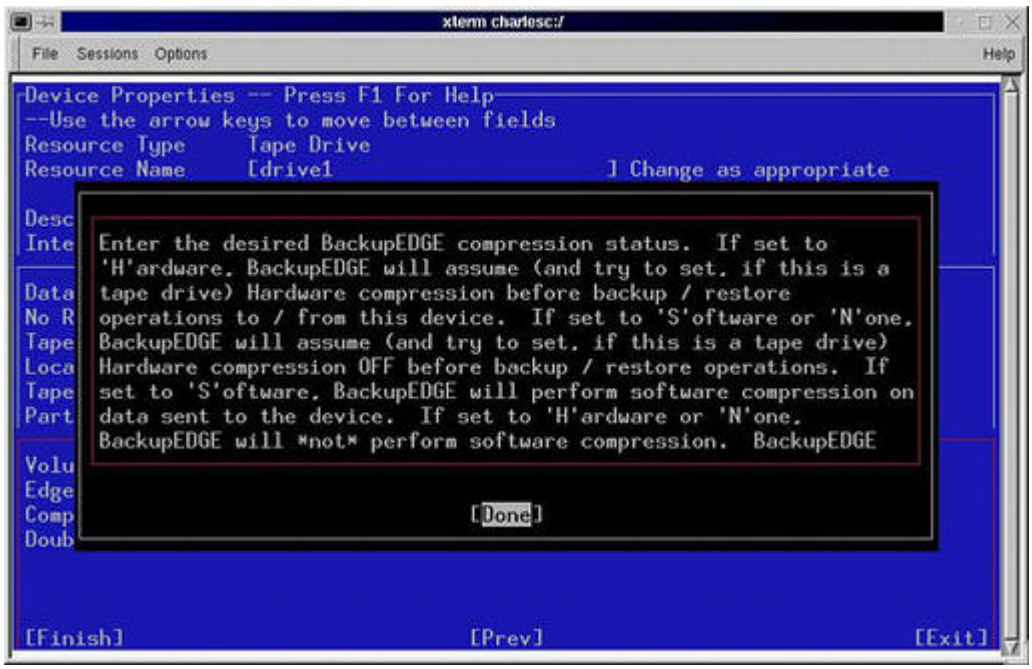


Figure 2. Installation Script with On-Line Help

The installation does an excellent job of determining what tape drives you have. It correctly identified both of my SCSI tape drives, probably by reading the /proc filesystem. The installation script also will detect whether your tape drive will do fast seeks and what the threshold for changing to a fast seek might be (see Figure 3). Fast seeks are great for restoring one or just a few files, a common restoration scenario. The installation is the best tape-drive characterization process I've seen on Linux so far.

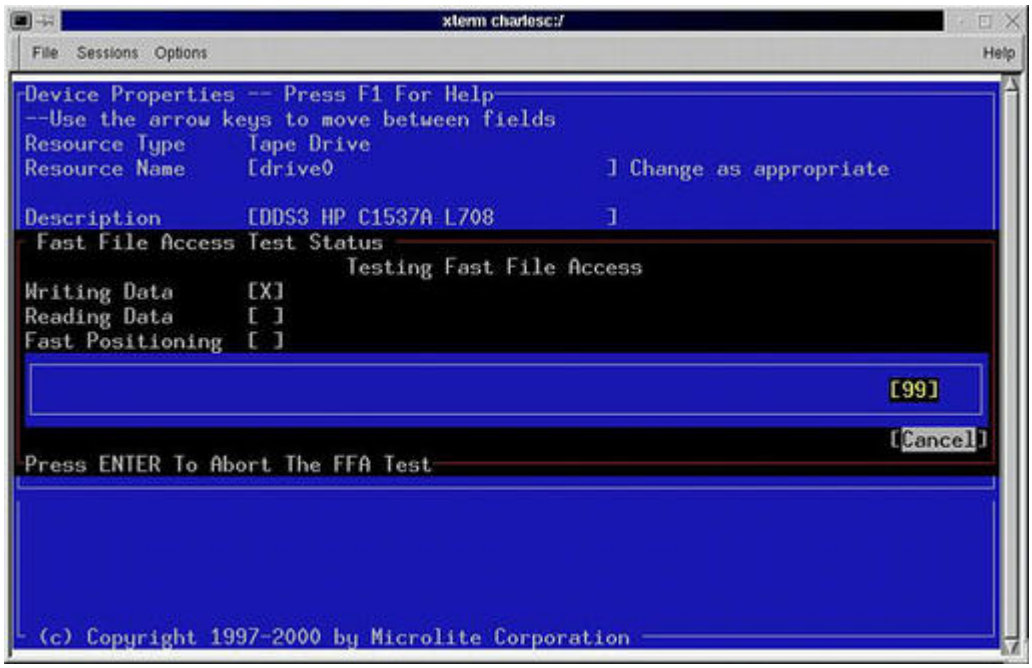


Figure 3. Installation Script Determining Fast Forward Threshold

The installation script will even set up a background task to check for sparse files (Microlite calls them “virtual files”). Correct handling of sparse files can save vast amounts of media on backup and even greater amounts on restore. For those systems that have sparse files, lack of proper sparse-file handling can rule out a backup product. Unfortunately, on both of my testbed computers, the search program failed with an error number but no real explanation why. Fortunately, there is a text file of sparse files you can hack, and Microlite documented doing so. BackupEDGE also supports raw filesystem partitions, useful for database servers.

The installation even put an icon on my KDE desktop. A simple hack to the shell script launched by the icon allowed me to fix the font size, a necessity for us geriatric penguinistas.

The Program

The first thing I did after installation was fire up the program, edgemenu, from the command line (see Figure 4). The program's color scheme, a blue background with gray characters, reminded me of Colorado Memory Systems DOS-based menus of ancient history. Your choices for color schemes appears to be gray on light blue or monochrome. Sometimes when exiting edgemenu, it leaves its color scheme on both the KDE konsole and xterm. Big deal, I can live with this for reliable backups.

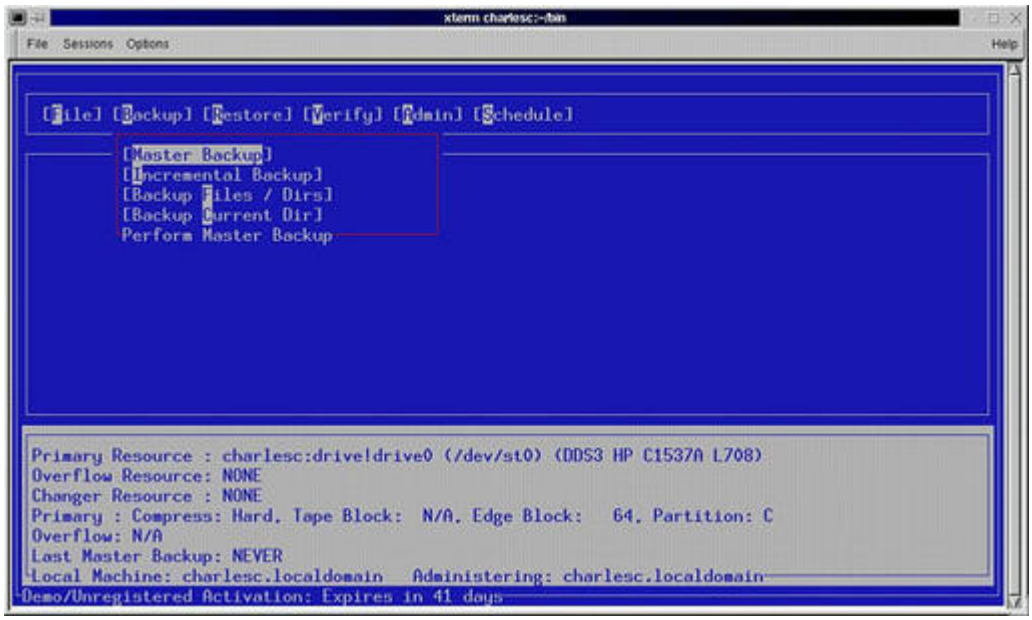


Figure 4. edgemenu

There is a full command-line capability, rather like tar, and “man edge” lists all the options available. Also, since the console-mode menu program is a front end for the command-line programs, you can study the commands it produces.

You can easily schedule automatic backups with the edgemenu program. It installs the backups into root's crontab, making it easy for you to adjust the backup in order to play with other cron jobs.

Backing Up

Naturally, the first thing I tried to do was a small test backup of about 9MB using my /etc directory. When I first tried this, the backup failed. Possibly the scan for sparse files affected the SCSI host adaptor. In any case, once I rebooted my system, I got successful backups on both of my tape drives.

The reason for the reboot was that I used another computer to test backups over the network, a process Microlite calls “remote backups”. I reconfigured BackupEDGE to use ssh because I already have ssh working with public key authentication, thus allowing secure transfers of data without passwords. I was able to configure the client machine and start a backup, but it locked up the server. Oops. After rebooting the server, I was able to back up on the server, but not on the client. My second attempt did not crash the server, which was an improvement. You also can use any host to administer another host. Apparently, the rsh or ssh connection is made and broken over the course of a session, so you may have waits while hosts authenticate.

Verification

Verifications can be done against the original file or by checking the CRC checksums stored with the data. The latter is useful for verifying a file after the original has changed. It is also a quick-and-dirty acceptance test of tape-drive head alignment, which is useful for less expensive tape drives, like some of the QIC offerings. Verification can be done as a routine part of the backup process, which is great.

Restoration

Restoration is easily done from the edgemenu program. You can restore redirected files to another location via any of the three interfaces. For the GUI addicts, there is an X-based restore tool, edge.emx (see Figure 5), that you may launch from edgemenu. It is suited for restoring individual files. Selecting a directory also selects the files and directories under it, if the directory is not expanded. The process is simple enough: select a database and click on it. Click your way down the tree until you select all the files you want. Click on Transfer to add the selected files to the restore window. Then click on Restore to restore the data.

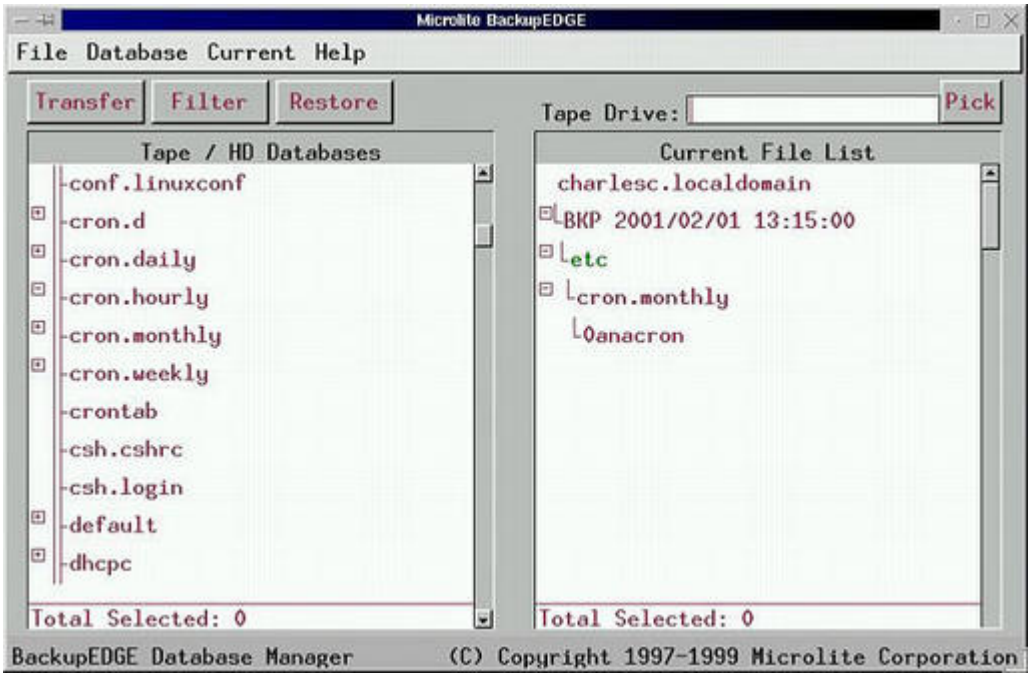


Figure 5. edge.emx

Documentation

The manual is large, over 270 pages. With reasonably large type and a fair amount of white space, it is easy to read. Seventy pages of the manual duplicates the man pages, and some of that is for operating systems other than Linux. Then there is another 130 pages of documentation on the crash recovery software. A good contents page is provided in each volume, but the index is a

bit sparse. For example, there is no entry for sparse files, and you have to already know that Microlite calls them “virtual files”. Overall, the documentation is plentiful, extensive, conversational and easy to read. Microlite gets an “A” here.

Part of the documentation expands on error messages. When a program produces a terse error message, you can look it up in the documentation and get a more detailed explanation. Other software vendors should learn to do this.

The documentation even tells you how to customize some aspects of BackupEDGE. For example, remote backups are done using rsh. However, the exact steps you need to take in order to use ssh are documented.

Support

Customer support is provided via e-mail, phone, fax or web site. There is no e-mail list that allows customers to exchange experiences directly. I did run into one problem that led me to customer support: I tried to substitute ssh for rsh. We never did get that to work, possibly because I took sick while trying to debug this problem. The support I got was polite but appeared to be perfunctory. E-mail responses were timely and had I not gotten sick, we probably would have gotten ssh working in time for this review.

Disaster Recovery

Disaster recovery should be very easy to do with RecoverEDGE, BackupEDGE's disaster recovery software, once you have it set up. I say “should be” only because I actually have not tested the restore process. Using HP's OBDR (basically a bootable tape drive) or a floppy disk set you build with BackupEDGE, you can make a backup tape for disaster recovery. When you need it, boot to the floppy or tape drive and away you go. The Microlite RecoverEDGE software will also adjust your partition sizes as needed in case you are restoring to a larger hard drive. Even machines that back up over the Net can use RecoverEDGE, which is more than you can do with OBDR.

Please note: the current version of this product was not available at the time of this writing.

Product Information/The Good/The Bad

Charles Curley (w3.trib.com/~ccurley“) is a freelance software engineer, writer and occasional cowpoke in the wilds of Wyoming.

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #90, October 2001

Readers sound off.

Lay off the Object Pascal

As a longtime Delphi developer I've been reading everything I can find about Kylix in anticipation of trying it out one of these days. However, I was disappointed in Petr Sorfa's review of the product (*LJ* August 2001 issue), in particular with his unsupported assertion that “the main problem, of course, is the Object Pascal programming language itself.”

If that's the main problem with the product, at least say why. As far as I can tell from the review his major problem with the language was “getting used to the := symbol for assigning variable values”. Big deal—all new languages take some getting used to.

Methinks there's some C++ snobbery here. Object Pascal is, admittedly, a proprietary language (like Java), which is what he may be getting at. However, it is a full-featured, object-oriented language similar to Java in features and general approach (with the exception of Java's automatic garbage collection). I think it deserves a little respect.

—Tom HavilandTHAVILAN@suss.com

Geek Disclaimer

I've been interested to read the Geek Law column in the last couple of issues of *LJ*. While I wasn't surprised to see a disclaimer at the end of each column, I was surprised that there's no mention of the advice being relevant only to the US legal system. While the articles so far have mentioned Congress and the US, I think it would be helpful for it to be stated explicitly that the author is writing from the point of US law.

Even in the UK (from where I am writing), we have three different judicial systems: one for England and Wales, one for Scotland and one for Northern Ireland. There's no guarantee that something that is legal in England and Wales is legal in Scotland, for example.

—Martin Radfordmartin@zamenhof.demon.co.uk

Who says one voice can't make a difference? Larry reminds us that not only does law vary nation to nation, but even within the US from state to state. He has modified the disclaimer.

—Editor

Ditch the French

I have to comment on the Cooking with Linux column by Marcel Gagné. The French chef gag is funny for about two minutes. Then it becomes tiresome. Another few minutes and it becomes downright irritating. By the time I finish a column, I'm ready to lock François in the wine cellar and throw away the key. So I seldom finish the column, which is a shame because otherwise it's well written and interesting. Please, Marcel, consider ditching the gag.

—Daniel D. Jonesetc-daniel.d.jones@cnet.navy.mil

Mon cher ami, perhaps the ungarnished and simple shipboard food you enjoy in the navy has influenced your palate. Some people like a little French seasoning with their technical staples. Linux isn't all free bière—un peu du vin est nécessaire aussi.

—Editor

Surprised but Unoffended

I am surprised that one of your readers should be offended by a biblical reference in the June 2001 Best of Technical Support, page 96. Perhaps Noel Moss has nothing better to do than to take it upon him/herself to try to curtail the author's first amendment right to freedom of the press, regardless of literary intent. I'm certain the reader would not have been offended if the reference had come from the *Koran*, or the *Book of the Dead*, rather than the *Bible*. I read *Linux Journal* cover to cover and have not once been offended by its content. Keep up the good work.

—Paul Barkerpdbarker@nevinslake.com

Thanks Paul, I was beginning to think there were no unoffended readers left.

—Editor

Searching for Open Kylix

The August 2001 issue of *Linux Journal* stated:

> There are three editions of Kylix: Open, which is available for free (downloadable) for noncommercial GPL development (or $99 US for a hard copy version); Developer, for commercial use with a limited number of features and components ($999); and Server, with all of the features and components ($1,999).

I have reviewed most of the Borland web site (including community.borland.com) and can't seem to locate any but the trial edition of Kylix for download. How can I find this open (GPL) edition for download?

—Brinkley Harrellbrinkley@fusemeister.com

Kylix Open Edition was released only recently and is now available at borland.com/kylix/openedition.

—Editor

Archive Index Issue Table of Contents

Advanced search

upFRONT

Various

Issue #90, October 2001

Stop the Presses, *LJ* Index and more.

Postscript to This Month's Geek Law Column

Shortly after I wrote this month's article I went to the O'Reilly Open Source Convention in San Diego. There, on a display table in the exhibit hall, I found cans of a beverage containing carbonated water, corn syrup, caramel color and caffeine, proudly bearing the trademark “Open Cola”. Five cents from every can of Open Cola will be donated to the Free Software Foundation. So it does matter which brand of cola you drink! By the way, the recipe for Open Cola is available under the GPL; see opencola.com.

LJ Index—October 2001

1. Total compensation in billions of dollars for the top executives at the top 807 companies in Silicon Valley in the last fiscal year: 4.8
2. Above number as a multiple of the prior year: 2
3. Percentage of decline in stock prices of the MN 150 Index, which tracks the largest Silicon Valley companies over the same period: 24
4. Number of times the word “shit” appears in the first “South Park” program of the latest season on Comedy Central, according to an odometer that displayed a running count on the screen: 142
5. Number of e-mails received by Comedy Central in response to the same “South Park” episode: 4
6. Percentage of received e-mails supportive of profanity in the episode: 100
7. Number of patents issued in the year 2000 by the United States Patent and Trademark Office: 158,118
8. Position of IBM among companies receiving US patents in 2000: 1
9. Number of US patents issued to IBM: 2,886

10. Number of US companies in the top ten recipients of US patents in 2000: 4
11. Number of Japanese companies in the same top ten: 6
12. Losses in millions of dollars by Webvan when it went Chapter 11 in July 2001: 860
13. Number of pages crawled by Google: 1,346,966,000
14. Number of Google-searched pages in which “sun” appears: 25,500,000
15. Number of Google-searched pages in which “microsoft” appears: 20,200,000
16. Number of Google-searched pages in which “dell” appears: 14,700,000
17. Number of Google-searched pages in which “solution” appears: 13,300,000
18. Number of Google-searched pages in which “ibm” appears: 11,200,000
19. Number of Google-searched pages in which “unix” appears: 10,900,000
20. Number of Google-searched pages in which “perl” appears: 7,650,000
21. Number of Google-searched pages in which “python” appears: 2,070,000
22. Number of Google-searched pages in which “linux” appears: 31,600,000
23. Linux-referenced pages per thousand Google finds on the Web: 2.35

Sources

- 1-3: *San Jose Mercury News*
- 4-6: *The New Yorker*
- 7-11: United States Patent and Trademark Office
- 12: *The Wall Street Journal*
- 13-23: Google, July 12, 2001

Windows Server Gains Appear to Be at Sun's Expense

Netcraft's July Web Server Survey (netcraft.com/survey) showed a huge jump in Microsoft IIS' share of web server software usage on 31,299,592 surveyed net-connected computers. After reaching a plateau of around 20% in 1998, IIS suddenly jumped nearly 5% to 25.88%. Apache reciprocally declined by 4.29% to 58.73%. Microsoft's gain represented about 2% of all active sites on the Web.

Netcraft attributed the gain to a single event: the conversion of domain registrar Namezero's servers from Solaris to Windows 2000 and from Apache to IIS, along with a related move by part of Network Solutions' domain registration system. Network Solutions also moved physically from Digex to Interland (where Microsoft has held a minority interest). “These large installations had previously been masking a general decline in Solaris share on the Web, which is now down four percentage points over the last year”,

Netcraft reported. “Additionally, the Network Solutions site was by far the largest Netscape-Enterprise installation in terms of numbers of hostnames, and one would expect that Netscape-Enterprise overall share will drop toward the 2-2.5% it has in the active sites analysis over the next few months.”

The previous month's survey also showed a shift in Windows' direction, again at Solaris' expense. In that survey, which attempted to count computers rather than hosts, Netcraft found that 49% of the surveyed computers were running Windows. Linux accounts for about 28%. And, all UNIX-related computers accounted for 45%. The remaining 6% were non-UNIX or unknown. “As some of the 3.6% of computers not identified by Netcraft operating system detector will in reality be Windows systems”, Netcraft reported, “it would be fair to say about half of public web servers world-wide are run on Microsoft operating systems.”

Netcraft also reported that Linux “has been consistently gaining share since this survey started but, interestingly, not significantly to Windows' detriment. Operating systems that have lost share have been Solaris and other proprietary operating systems, and to a small degree BSD.”

The significant interpretation of the data, Netcraft suggests, is that Solaris is “being continually chased further and further up market by Intel-based operating systems, with Sun in turn progressively eliminating the other proprietary UNIX operating systems.”

—Doc Searls

OpenPGP Signatures: the New Baseball Cards?

How well-connected are you? Drew Streib can tell you to four decimal places. Drew, who now runs an OpenPGP keyserver in addition to his other thankless tasks, is currently publishing monthly reports on how closely OpenPGP users are connected to the Web of Trust. His math, based on earlier calculations by Neal McBurnett, is complicated, but the result is a current map of the community's Web of Trust.

Closest to the center of the Web are crypto luminaries and organizers of keys-signing events, including Peter N. Wan, Ingmar Camphausen and Theodore Ts'o. Philip R. Zimmermann, who wrote the original PGP, is only number 24.

Drew's report comes at an exciting time for encrypted mail. GNU Privacy Guard, a free OpenPGP implementation, is available in common distributions, support in popular mailers such as mutt makes encryption convenient to use and the FBI's much-publicized Carnivore snooping system certainly hasn't hurt.

Signing people's keys to do better in Drew's rankings might seem like a pointless game, but it really does expand the Web of Trust. You can never lose juice by exchanging signatures with someone else, and it helps everyone's ability to send trusted, encrypted mail. Even if you sign the key of some “lamer” at the bottom of the list, you'll both move up next month. (As for me, I got a Theodore Ts'o! Look out next month.)

Get started with PGP using the free implementation, GNU Privacy Guard, from gnupg.org. Then read Drew's report at dtype.org/keyanalyze.

—Don Marti

Stop the Presses: Something to CARP about

On Monday, July 30, 2001 the US Copyright Office convened the Copyright Arbitration Royalty Panel (yes, CARP, loc.gov/copyright/carp) to make a decision shortly on conditions under which webcasters will be required to make royalty payments. The results could be highly inconvenient for webcasters of all kinds. Howard Greenstein, a webcasting pioneer, puts it this way in his weblog:

> Webcasters, many of whom have been accounting for what they have estimated they would have to pay under a negotiated compulsory license (and putting aside revenue for years) are about to find out (within 60 days) what it will cost them. Unless, of course, they are an “interactive” station. If you're a standard station under the Digital Millenium Copyright Act, you play music in a certain way. You don't give people much choice about what they hear.

Yet the number of streaming sources on the Net runs into uncounted thousands (or perhaps millions). What's more, many of these are far more interactive than traditional broadcasting has ever been or can even comprehend. What's the news for them? Easy: work outside the system.

That's what KPIG has been doing since it became the first commercial radio station ever to broadcast on the Web. KPIG broadcasts from (no kidding) Freedom, California on 107-oink-5 on the FM band. On the Web, however, KPIG is a virtual Idaho. Its 128KB MP3 stream is one of the Web's hi-fi music beacons. So are the half-dozen or so other streams the station puts out at various speeds for various clients and bandwidths (and with content other than KPIG alone). Naturally (their site reports) they digitize that content on a Linux PC with an open-source LAME MP3 encoder (mp3dev.org/mp3).

KPIG, which once described its format as “mutant cowboy rock and roll”, is one of the few remaining commercial stations where the disc jockeys still choose the music, and community ties are so close it's hard to tell where the station

ends and its constituency begins. As a successful business (it has always done pretty well in the ratings and sells plenty of advertising), KPIG also has managed to remain both artist- and industry-friendly. Every song the station plays is listed live on the Web, along with links that make it easy to buy the CD, research the artist or follow a tour schedule. Without a doubt, KPIG owns the high-mud mark for combining commercial success, community involvement, resourceful use of free and open-source software and adaptiveness to a surreally perverse environment.

The hacker in chief at KPIG is “Wild Bill” Goldsmith, one of KPIG's Founding Farmers and the proprietor of RadioParadise.com. Unencumbered by the need to participate in the fully regulated environment of commercial broadcasting, Radio Paradise is beating a path through the uncharted wilderness where artists and technically smart connoisseurs will rebuild their own industry from the outside in. Asked for the technical angle on Radio Paradise, Bill writes:

> [Radio Paradise is] based on a set of software tools—for picking and scheduling music and doing voice tracks from anywhere over the Net, and for accepting and organizing listener feedback on my playlist. Everything I'm doing software-wise is 100% open source: Linux, PHP, Perl, Postgres, and Icecast.
>
> I am convinced that what you see at radioparadise.com represents the future of radio, or of quality radio, anyway: very interactive, tightly controlled artistically (no random segues, everything happens for a reason)--completely free from the influences of the radio/music industry hype machine (to the best of my ability, anyway)--and supported primarily by voluntary contributions from listeners.
>
> This isn't a game plan that's going to make anyone rich. But it can make it possible for anyone with talent to make a very comfortable living without compromizing their integrity in any way—and that's all I for one have ever wanted.

I'm an old radio freak and have been a fan of KPIG and its ancestors going back to the Sixties. Living, breathing radio stations like KPIG, run by people who love the business more for the good it does than for the money it makes, have gone out like candles in the rain—first one by one, then by the dozens and finally by the thousands.

It's not surprising to find a hacker starting a bonfire with the last candle that stands.

—Doc Searls

They Said It

Sober nations have all at once become desperate gamblers and risked almost their existence upon the turn of a piece of paper. To trace the history of the most prominent of these delusions is the object of the present pages. Men, it has been well said, think in herds; it will be seen that they go mad in herds, while they only recover their senses slowly, and one by one.

—Charles MacKay, 1841

You can't wake a person who is pretending to be asleep.

—Navajo Proverb

The artistic temperament is a disease that affects amateurs.

—G. K. Chesterton

Jetlag is evil. But not as evil as Flash.

—Deborah Branscum

Friends help you move. Real friends help you move bodies.

—polar bear on Slashdot

One night I was layin' down, I heard mama 'n papa talkin', I heard papa tell mama, let that boy boogie-woogie, it's in him, and it got to come out. And I felt so good, went on boogie'n just the same.

—John Lee Hooker

To suggest that the author knows best how to write effectively to each individual reader is silly, yet that's what I understand of your position.

—John Wilcox, Microsoft employee, defending Smart Tags

Even if Smart Tags don't violate copyright or deceptive trade laws, they still violate the integrity of the Web. Part of the appeal of the Web is that it allows anyone to publish anything, to take their thoughts, feelings and opinions and put them before the world with no censors or marketroids in the way. By adding Smart Tags to web pages, Microsoft is interposing itself between authors and their audience. Microsoft told Walter Mossberg, “The feature will spare users from under-linked sites.” Microsoft is in effect deciding how authors should write, and how developers should build, web sites.

—Chris Kaminski

Intellectual property (IP) has been driving the species for some five million years. In the past 100 or so years, it's increasingly been saddled with the chore of lining the pockets of middlemen and parasites who, sans this lining, would lack sufficient intellect to open a can of beer.

—Tom Matrullo

The genius of you Americans is that you never make clear-cut stupid moves, only complicated stupid moves that make us wonder at the possibility that there may be something to them that we are missing.

—Gamel Abdul Nasser

If the business notion of best practices had been applied from the dawn of human civilization, human beings never would have achieved civilization. Art history would focus on things like ancient Roman bas-reliefs of the current Tide and Cheer equivalents, the Sistine Chapel ceiling would say “Bank With Medici!” and instead of a torch, the Statue of Liberty would be brandishing a tube of Preparation H.

—Christopher Locke

We are natural villagers. For most of mankind's history we have lived in very small communities in which we knew everybody and everybody knew us. But gradually there grew to be far too many of us, and our communities became too large and disparate for us to be able to feel a part of them, and our technologies were unequal to the task of drawing us together. But that is changing.

—Douglas Adams

Archive Index Issue Table of Contents

Advanced search

Brains, Not Trains

Richard Vernon

Issue #90, October 2001

Artificial Intelligence: where will it end? Not at the movies!

This month's focus on engineering turned out to be pretty exciting, what with articles on wind tunnels and lasers. We thought we'd put some nice destructive lasers burning through steel (or space shuttles) on our cover, but Brian Gollsneider at the Army Research Lab at Adelphi, Maryland informed me that the lasers with which they work have an 850nm wavelength intended for communication purposes only and are outside the range of human visibility. Being a “journal”, we like to keep our covers realistic and article-related (remember the little man inside the computer on the February 2001 issue?). So we went with the skier.

As Marcel Gagné reminds us in his column this month, engineers have traditionally been conceived as people who drive trains (hence the notion of chemical engineers as those who take drugs and drive trains). But as this month's articles show, most engineers seem to have abandoned internal combustion and moved to other types of engines. Rick Lehrbaum reveals Isamu, a robot with remarkably humanoid abilities for movement. The project is a joint venture between the University of Tokyo's Jouhou System Kougaku (JSK) Laboratory and the Aircraft and Mechanical Systems Division of Kawada Industries, Inc. Professor Hirochika Inoue heads the JSK Lab, and his views are among those of the many roboticists featured in the book, *Robo-Sapiens*, which documents current robotic projects around the world and speculates as to the future of robo-human relations.

There seems to be three schools of thought among roboticists concerning this future: robots will surpass humans in intelligence and ability, robots will never approach humans or humans themselves will become increasingly robotic.

One particularly impressive project highlighted in the book is DB (Dynamic Brain) of the Kawato Dynamic Brain Project at Advanced Telecommunications

Research Institute near Kyoto, Japan. The DB robot is used by neurophysicists who work with the robot to learn more about the functioning of the human brain. The scary thing about DB is it learns, not through programming, but by watching and mimicking the movements of humans. It's already learned to balance objects and juggle better than its instructors. There is one glimmer of hope however, at the book's press time, it still couldn't dance very well.

If the future should bring robots superior to humans, let's hope they run Linux—perhaps running open-source software will make them more inclined to share the earth with their inferior humanoid cousins.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #90, October 2001

Our experts answer your technical questions.

Really Big malloc()

I would like to understand what the issues are that limit the memory available to a single process under Linux.

I have an Athlon 1.2GHz processor with 1.5GB of RAM and 2GB of swap space. The OS is Red Hat 7.1 with kernel 2.4.3-12. The system sees all 1.5GB on bootup and reports as much through top and other utilities. But alas, single processes can grab no more than roughly 940MB. I'm testing the process memory limit by running a simple C program that allocates a single large char array.

—Ned Piburn, npiburn@oti.gd-ots.com

The Linux kernel has a setting on how the memory is split between the kernel and user space. It may be that your specific kernel is built to give 3GB to the kernel and 1G to user space. When you go in the kernel configuration (**cd /usr/src/linux; make menuconfig**), check “Processor type and features / Maximum Virtual Memory” and set it to 3GB (some patched kernels have bugs if you use 2GB).

—Marc Merlin, marc_bts@valinux.com

GNU libc uses brk() for small allocations and mmap() for larger allocations, and only about 900MB can be allocated using brk(). Many small allocations might fail where fewer large allocations would succeed. If this is the problem your code is running into, one solution would be to write a custom malloc()--either one that always allocates memory using mmap() or one that first mmap()s large chunks and then parcels out fragments within the chunks.

—Scott Maxwell, maxwell@ScottMaxwell.org

The mallopt function is covered in the GNU Info documentation for libc under “Malloc Tunable Parameters”. Set M_MMAP_THRESHOLD to force malloc() to use mmap() instead of brk().

—Don Marti, info@linuxjournal.com

Where's My Fourth CPU?

I've installed SuSE 7.1 on a three-processor machine and then added a fourth processor. The OS doesn't see the new CPU. It's a Compaq dl580, x86 architecture. Do you have to recompile or do something extra to get the OS to acknowledge the new CPU?

—Chet Jaynes, cjorlb@pacbell.net

Linux should be able to detect and use the fourth processor as long as the motherboard does. Use the BIOS setup in order to make sure the system itself is enabling it.

—Mario Neto, mneto@argo.com.br

RPM Can't Upgrade RPM

I'm currently using Red Hat Linux 6.2 with RPM-3.0.3. To upgrade from RPM-3.0.3 to RPM-4.0.2 I tried to install db3-3.1.17 as prescribed but got the error message:

```
rpm can only install packages with
major version number <= 3
```

—Atul, atul_info@yahoo.com

Install the latest release of version 3 RPM, as that deals with both RPM3 and RPM4. You can get it from ftp.rpm.prg/pub.

—Keith Trollope, keith@wishing-well.demon.co.uk

Adaptec SCSI Card under Red Hat 6.2

My Red Hat 6.2 automatically detects the Adaptec 29160 card, and a dynamic module AIC-7xxx is added to /etc/conf.modules. But when I connect an SCSI hard drive to the card, there is no /dev/sda available for fdisk. The device file exists but cannot be accessed by fdisk. If I boot Red Hat 7.1, the SCSI disk is recognized and works. But I need to boot Red Hat 6.2 with this SCSI card. How can I make it work under Rh6.2?

—Joshua, cschen@asiaa.sinica.edu.tw

One solution would be to install your Red Hat 7.1 kernel on a Red Hat 6.2 distribution. You will also need to upgrade a few other packages like modutils.

—Marc Merlin, marc_bts@valinux.com

I had the same problem when I got my Adaptec 29160 (great card, by the way). I installed Linux on an IDE drive temporarily, got a recent 2.2 series kernel from a kernel.org mirror and built the AIC-7xxx driver into the kernel, not as a module. Then I rebooted with the new kernel and copied everything over to the SCSI drive.

—Don Marti, info@linuxjournal.com

Bad User! No “cd ..”!

What can I do to make a user's directory be like a root directory, where the user just has an access to that directory or subdirectory?

—Rafael, rafaelss@ig.com.br

What you need is chroot. Many FTP dæmons chroot by default. If you want Telnet (or even better, SSH) to chroot, you can make a chroot shell. For more information go to freshmeat.net/projects/jail_c.

—Ben Ford, ben@kalifornia.com

Once you jail a user in, let's say, /home/user, you'll have to make some portion of /lib and /bin available under /home/user if you want the chrooted user to be able to run any commands at all.

—Marc Merlin, marc_bts@valinux.com

I Have No DNS and I Must wvdial.

I'm trying to connect to my ISP using an external terminal adaptor. But when wvdial connects I receive the following errors:

```
--> warning, can't find address for 'suse.de'
--> warning, address lookup does not work
--> Nameserver (DNS) failure, the connection may
    not work
```

—Mitko, mitak@post.com

Check in /etc/resolv.conf and make sure you have lines with:

```
nameserver aaa.bbb.ccc.ddd
```

where aaa.bbb.ccc.ddd is the IP addresses of a working DNS server. You can also look at the PPP log in /var/log for hints on where the problem lies.

—Felipe E. Barousse Boué, fbarousse@piensa.com

Archive Index Issue Table of Contents

Advanced search

New Products

Heather Mead

Issue #90, October 2001

Recital Linux Developer, Cleanscape SourceMill, TurboLinux for IBM eServer and more.

Recital Linux Developer

The Recital Linux Developer is a complete multi-user database, 4GL and suite of tools for developing and deploying Linux character mode applications. The package provides language and data compatibility with FoxPro, FoxBASE and Clipper, allowing migration of existing applications. The RAD environment has an integrated database, is Java-enabled and provides support for POP3, SMTP and import/export XML. Access to Recital, FoxPro, dBase, Informix and DB2 data is handled through standard xBase command syntax and user-designed forms.

Contact Recital Corporation, Inc., 85 Constitution Lane, Danvers, Massachusetts 01923, 800-873-7443 (toll-free), info@recital.com, www.recital.com.

Cleanscape SourceMill

Cleanscape Software announced the availability of Cleanscape SourceMill, an automatic source-code generation tool designed to aid rapid development and application modification by automating redundant programming tasks. Commercial-grade code is generated from object models and code patterns. Software developers can create application frameworks for multiple target environments by using SourceMill to instantiate application designs with templates. SourceMill also can be used to control consistency and enforce programming standards.

Contact Cleanscape Software International, 2231 Mora Drive, Suite E, Mountain View, California 94040, 650-864-9600, sales@cleanscape.net, cleanscape.net.

TurboLinux for IBM eServer

TurboLinux 6.5 is a fully supported distribution for the IBM eServer iSeries and pSeries, systems designed for the small- to mid-sized business. TL 6.5 provides firewall and web server capabilities, along with compatible file, print and e-mail services. The unified code base simplifies global deployment through support for Li18nux and LSB standards. A single iSeries server can support up to 31 separate Linux servers. Each Linux server runs in its own partition and is able to share processors, disk, tape, CD-ROM, DVD and LAN resources with the other applications running on the iSeries server.

Contact TurboLinux, 8000 Marina Boulevard, Suite 300, Brisbane, California 94005, 650-228-5000, turbolinux.com

Black Adder 1.0beta3

The beta3 release of Black Adder 1.0, a Linux/Windows UI development environment for Python and Ruby based on Qt, is now available. Black Adder combines a visual design environment with debugging, syntax highlighting, ODBC interfaces and extensive HTML documentation into a comprehensive platform for developing Python and Ruby applications. Changes for this release include Qt v.2.3.1 and support for Python v2.0.x and 2.1.x, among others.

Contact theKompany.com, PO Box 80265, Rancho Santa Margarita, California 92688, 949-713-3276, sales@thekompany.com, thekompany.com

FlexeLint for C/C++

Version 8.0 of FlexeLint for C/C++ is now available. FlexeLint is a static analyzer that will analyze a mixed suite of C and C++ programs and report on bugs, glitches and inconsistencies to help develop maintainable and portable programs. New to version 8.0 are interfunction value tracking, improved exception handling, checks for adherence to MISRA guidelines and 20 new options. Other checks include user-defined function semantic checking, pointer tracking and control flow-based analysis of variable initialization.

Contact Gimpel Software, 3207 Hogarth Lane, Collegeville, Pennsylvania 19426, 610-584-4261, sales@gimpel.com, gimpel.com

StuffIt Engine SDK

Aladdin Systems offers the StuffIt Engine Software Developer Kit, allowing developers to integrate compression into their projects. The SDK can be used on the fly to wrap a group of files into a single self-extracting archive for the user's computer or to reduce the time it takes to send and receive files.

Supported compression and encoding formats are StuffIt, Zip, Gzip, Tar, Rar, Bzip2, UUencode, UNIX Compress, BinHex, MacBinary and more, and all formats are accessible from a single API.

Contact Aladdin Systems, Inc., 245 Westridge Drive, Watsonville, California 95076, 831-761-6200, info@aladdinsys.com, aladdinsys.com

Beowulf Professional Edition

Scyld Computing released the Beowulf Professional Edition, cluster operating system software that is ready to run out of the box. The Professional Edition offers simplified cluster setup, integration and administration, along with documentation and support from the original Beowulf development team. New additions to this version include full Alpha support, Myrinet and Gigabit Ethernet support, the batch queue system (BBQ), web-based administration and job monitoring, and PVFS, NFSv3 and ROMIO filesystems.

Contact Scyld Computing Corporation, 410 Severn Avenue, Suite 210, Annapolis, Maryland 21403, 410-990-9993, sales@scyld.com, scyld.com

CapeStudio RAD Tool

The CapeStudio RAD web services development tool enables software developers to build and deploy web services. CapeStudio automatically generates code for Java or Visual Basic from web services description language (WSDL) files, which describe the interfaces to web services, reducing the time needed to build the service. A graphical environment for defining bidirectional transformations between XML and SOAP messages is also included in CapeStudio. It is a standalone environment that works with any web services platform that adheres to the XML, SOAP, WSDL and UDDI industry standards.

Contact Cape Clear Software, Inc., 900 East Hamilton Avenue, Suite 100, Campbell, California 95008, 866-227-3226 (toll-free), info@capeclear.com, capeclear.com.

Archive Index Issue Table of Contents

Advanced search

Show Report, Day One

Doc Searls

Issue #90, October 2001

Doc goes to the show.

Back in May, Craig Mundie, Microsoft's Senior VP Advanced Strategies, made a very strategic move; he gave a speech at NYU's Stern School of Business that announced the terms by which Microsoft was cracking open—barely—its source code. He called the company's new licensing model "shared source". (I just wrote "scared source" by mistake, which tells you where my mind is headed.)

The fact that Microsoft would start rapping about *any* kind of source code, and modify it with a fresh new euphemism—*shared*—caused immediate tissue rejection in the hacker cultural body. Leading hackers so certain of their own Truth that they refuse to appear on each other's t-shirts were suddenly gathered around their collective keyboards to craft a single response that would say, in polite terms, "Embrace and extend *this*, dude."

The result was an open letter published on Bruce Perens' site, and signed by Bruce and a quotariat of Free Software and Open Source luminaries: Richard Stallman, Eric S. Raymond, Guido Van Rossum, Tim O'Reilly, Larry Augustin, Bob Young, Larry Wall, Miguel de Icaza and Linus Torvalds. (An anonymous coward on Slashdot wrote, "It's like a human Beowulf cluster!") While critical and challenging to Microsoft, its bottom line was open and inviting:

We urge Microsoft to go the rest of the way in embracing the Open Source software development paradigm. Stop asking for one-way sharing, and accept the responsibility to share and share alike that comes with the benefits of Open Source. Acknowledge that it is compatible with business.

Free Software is a great way to build a common foundation of software that encourages innovation and fair competition. Microsoft, it's time for you to join us.

Mundie came back with a piece in CNET that framed his argument in terms of economics, manufacture and the PC's popularity:

... this is more than just an academic debate. The commercial software industry is a significant driver of our global economy. It employs 1.35 million people and produces $175 billion in worldwide revenues annually (sources: BSA, IDC).

The business model for commercial software has a proven track record and is a key engine of economic growth for many countries. It has boosted productivity and efficiency in almost every sector of the economy, as businesses and individuals have enjoyed the wealth of tools, information and other activities made possible in the PC era.

Then he took on the GPL, the Free Software Foundation's General Public License:

In my speech, I did not question the right of the open-source software model to compete in the marketplace. The issue at hand is choice; companies and individuals should be able to choose either model, and we support this right. I did call out what I believe is a real problem in the licensing model that many open-source software products employ: the General Public License.

The GPL turns our existing concepts of intellectual property rights on their heads. Some of the tension I see between the GPL and strong business models is by design, and some of it is caused simply because there remains a high level of legal uncertainty around the GPL—uncertainty that translates into business risk.

In my opinion, the GPL is intended to build a strong software community at the expense of a strong commercial software business model. That's why Linus Torvalds said last week that “Linux is never really going to be a rich sell.”

This isn't to say that some companies won't find a business plan that can make money releasing products under the GPL. We have yet to see such companies emerge, but perhaps some will.

He added,

What is at issue with the GPL? In a nutshell, it debases the currency of the ideas and labor that transform great ideas into great products.

It would be easy to dismiss all this as provocation in the voice of boilerplate, or worse, “a declaration of war on our culture”, as one überhacker privately called it. But neither of those responses are useful to folks caught in the middle—the IT professionals my *Linux Journal* column calls “suits”.

As it happened Eric Raymond and I were both guests on the May 14 broadcast of "The Linux Show". When conversation came around to the reasoning behind open-source rhetoric, Eric said this:

We used the term open source not to piss off the FSF folks, but to claim a semantic space where we could talk about issues without scaring away the people whose beliefs we wanted to change.

So far this has been an extremely successful strategy for which Eric and the Open Source community deserve extreme credit. Even if IT suits don't agree about what "open source" means, they're all at least talking about it. And using lots of it, too.

But if markets are conversations, everybody who inhabits the open-source semantic space is involved in the market—a market that now includes Microsoft. What happens if Microsoft makes more sense to their constituency than we do? This is an important question. Conversing is not the same as believing. Remember Eric's last seven words from above. Most of the suits talking about open source are still *people whose beliefs we want to change*.

Changing other people's beliefs isn't like changing your shoes: it's like changing *other* people's shoes. Even if the other guy's shoes are ugly and uncomfortable, they're still familiar to him. And in this case, familiar doesn't cover it. In the IT world, Microsoft's platforms, software and tools are the prevailing environment that Microsoft wants very much to protect.

How much? Here's Steve Ballmer talking about Linux in January: "I think you have to rate competitors that threaten your core higher than you rate competitors where you're trying to take from them." He adds: "It puts the Linux phenomenon and the Unix phenomenon at the top of the list. I'd put the Linux phenomenon really as threat No. 1." He goes on to say Sun and Oracle are "second tier" rivals, adding, "I'd put AOL probably maybe at that level or a half-step down."

So how well does Steve Ballmer understand this threat? Not very, judging from this this wild conflation of open source, Linux and licensing:

The only thing we have a problem with is when the government funds open-source work. Government funding should be for work that is available to everybody. Open source is not available to commercial companies. The way the license is written, if you use any open-source software, you have to make the rest of your software open source. If the government wants to put something in the public domain, it should. Linux is not in the public domain. Linux is a cancer

that attaches itself in an intellectual property sense to everything it touches. That's the way that the license works.

Never one to shy from a fight, Eric Raymond blasted back a Q&A titled The Big Lie:

Other open-source licenses—such as the BSD license in the TCP/IP stack that Microsoft adapted for Windows—will never infect anybody's code or data, because they're designed not to. But Ballmer wants business people and the public to fear them all, because only if open source is general is discredited will Microsoft maintain its monopoly.

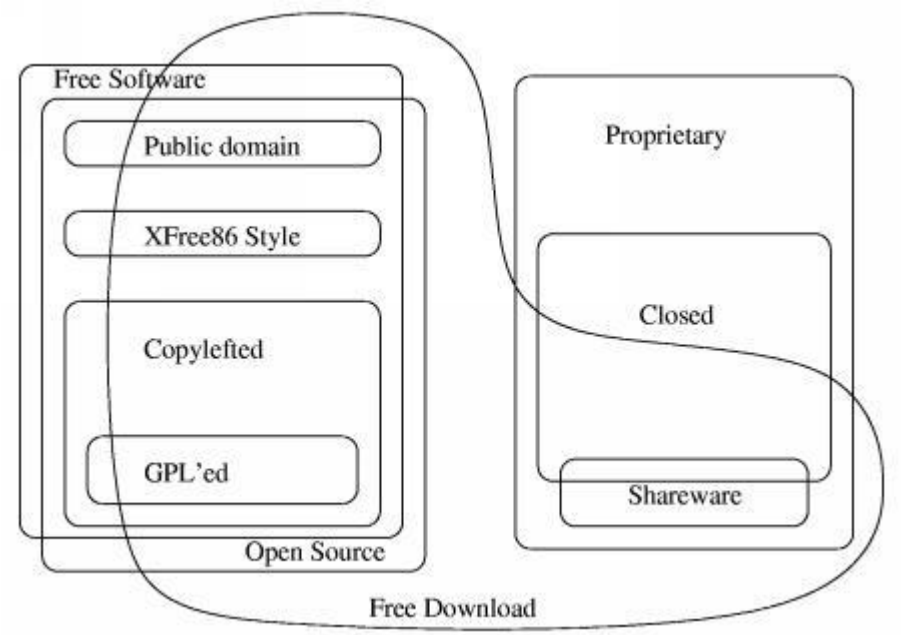
The Big Lie is a term originally coined to describe a characteristic form of Nazi (and later Soviet) propaganda. The essence of the Big Lie propaganda technique is that if you repeat the lie often enough over enough channels, people will soak it up through their pores and come to believe it as something “everybody knows”.

In the last three months, Jim Allchin and Craig Mundie and Steve Ballmer have launched a classic Big Lie campaign against open source. They have described it as “un-American”, “a destroyer”, and “a cancer”. They have deliberately confused the GPL with non-infectious open-source licenses, and they have deliberately confused active combination of code with passive aggregation of data. They have lied, and lied, and lied again.

Why? Because the most truthful thing Ballmer admitted in that interview is that yeah, Linux *is* a threat to Microsoft.

There's a good reason Microsoft can get away with a *lie & confuse* strategy: its #1 threat is pretty damn confusing already, without any help from Microsoft. Most confusing of all, from the perspective of common business sense, is the GPL, about which the Open Source community is both respectful and protective, even though there is plenty of disagreement about it. By hitting the GPL squarely where it appears least useful for business, Mundie disperses the community like a rack of pool balls. Suddenly they're all over the place, talking about all kinds of stuff.

To see what I'm talking bout, check out this diagram from the Free Software Foundation's philosophy page: fsf



From the perspective of both Microsoft and its customers, the one thing that's easy to understand (not right or wrong, just easy) is in the upper right. This is the familiar stuff that IT suits have been paying for since the Nixon administration. By aiming insults at the GPL in the lower left, Microsoft hopes everybody in the Free/Open communities will rush to defend what business folks have the most trouble understanding: the FSF's belief that software should not be owned.

In other words, Microsoft wants us to join a “debate” in which we defend the one thing we can't stop arguing about amongst ourselves. What's more, they've located their position—shared source—in a location toward which the business end of the open source world is headed anyway.

That's right. “Everybody's headed toward a hybrid model”, Larry Augustin told me last week. He didn't have time to explain exactly what he meant, but last night at dinner Eric Raymond gave me an explanation that included this interesting fact: VA will sell some closed-source software. He hedged with some pretty big qualifiers, but the word “closed” passed his lips.

A few days ago I asked another CEO what's going on. He said, “Call it `shared source,' `gated source' `source under glass'... we're all working in the same direction.” When I pressed him for a reason, he said, “We need to make money.”

Nobody in the commercial end of the Open Source community is eager to speak in vivid terms about the drift in their commercial source code policies—at least not yet. Meanwhile, Microsoft is very eager to do exactly that, giving them a certain advantage.

Will Craig Mundie will use that advantage on Thursday when he debates Red Hat CTO Michael Tiemann? Will open-source luminaries (a superset of the Beowulf Cluster who signed the original response to Mundie) take up the issue when I raise it with them at the Open Source Summit tomorrow? And will the open-source conversation grow to include independent commercial developers like Dave Winer and his company, Userland, which are doing open-source work (in Userland's case, on XML-RPC and SOAP) but have not been members of the community until now?

Finding out more about this stuff is Job One in San Diego this week. Stay tuned.

Doc Searls (info@linuxjournal.com) is Senior Editor of *Linux Journal*. His monthly column is Linux For Suits. He is also a co-author of The Cluetrain Manifesto.

Archive Index Issue Table of Contents

Advanced search