Advanced search

Linux Journal Issue #104/December 2002

Understanding Apple iPod Playlists
LINUX JOURNAL
VariCAD: Professional CAD for Linux
OpenLDAP Everywhere
HIGH-AVAILABILITY LDAP CLUSTERS with Heartbeat
OpenLDAP FOR MAIL, SAMBA and more at Midwest Tool & Die
THE NEW SERIAL DRIVER LAYER for the 2.5 kernel
UNDERSTANDING TREES in ReiserFS
HELP! My boss made me install warez!

Features

Highly Available LDAP *by Cliff White and Jay D. Allen and Cliff White*
You can have uninterrupted LDAP service, using freely available software.
Process Accounting *by Keith Gilbertson*
Here's a way the kernel and some simple utilities work together to track processes and help you find performance and security issues.
OpenLDAP Everywhere *by Craig Swanson and Matt Lung*
A single company-wide directory service offers mail address lookup and file sharing to Linux and Windows users.

Indepth

Playing with ptrace, Part II *by Pradeep Padala*
In part two of our series on ptrace, find out how to set breakpoints and change the code of a running process on the fly.
Linux Powers Four-Wall 3-D Display *by Douglas B. Maxwell*
With the aid of a custom video switcher, a Linux cluster beats an expensive proprietary UNIX system for high-end virtual reality.
Learning the iTunesDB File Format *by Patrick Crosby*
iPods aren't just for people who use computers from Mattel, no wait, Apple. Here's the playlist format. Don't all buy iPods at once, folks.

Embedded

Driving Me Nuts The Serial Device Layer *by Greg Kroah-Hartman*

Toolbox

Kernel Korner Trees in the Reiser4 Filesystem, Part I *by Hans Reiser*
At the Forge Creating OpenACS Packages *by Reuven M. Lerner*
Cooking with Linux A Process Smorgasbord *by Marcel Gagné*
Paranoid Penguin Configuring and Using an FTP Proxy *by Mick Bauer*

Columns

Focus on Software On System Administrators *by David A. Bandel*
IAAL: The Ethical System Administrator *by Lawrence Rosen*
Linux for Suits Identity as Business Opportunity? *by Doc Searls*

Reviews

VariCAD 8.2-02 *by Michael Baxter*

Departments

Letters
upFRONT
From the Editor
On the Web
Best of Technical Support
New Products

Archive Index

Advanced search

Highly Available LDAP

Jay D. Allen

Cliff White

Issue #104, December 2002

Creating a highly available authentication server using open-source software.

As an organization adds applications and services, centralizing authentication and password services can increase security and decrease administrative and developer headaches. However, consolidating any service on a single server creates reliability concerns. High availability is especially critical for enterprise authentication services, because in many cases the entire enterprise comes to a stop when authentication stops working.

This article describes one method of creating a reliable authentication server cluster. We use an LDAP (Lightweight Directory Access Protocol) server to provide authentication services that can be subscribed to by various applications. To provide a highly available LDAP server, we use the heartbeat package from the Linux-HA initiative (www.linux-ha.org).

LDAP Background

We are using the OpenLDAP package (www.openldap.org), which is part of several Linux distributions, including Red Hat 7.1. Version 2.0.9 ships with Red Hat 7.1, and the current download version (as of this writing) is 2.0.11. The OpenLDAP Foundation was created as "a collaborative effort to develop a robust, commercial-grade, fully featured and open-source LDAP suite of applications and development tools" (from www.openldap.org). OpenLDAP version 1.0 was released in August 1998. The current major version is 2.0, which was released at the end of August 2000 and adds LDAPv3 support.

LDAP, like any good network service, is designed to run across multiple servers. LDAP uses two major features: replication and referral. The referral mechanism lets you split the LDAP namespace across multiple servers and arrange LDAP servers in a hierarchy. LDAP allows only one master server for a particular directory namespace (see Figure 1).

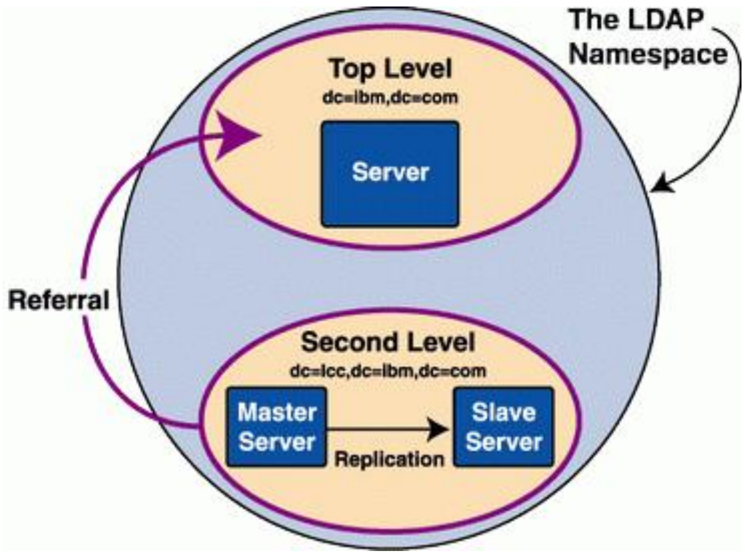


Figure 1. LDAP allows one master server per namespace.

Replication is driven by the OpenLDAP replication dæmon, slurpd, which periodically wakes up and checks a log file on the master for any updates. The updates are then pushed to the slave servers. Read requests can be answered by either server; updates can be performed only on the master. Update requests to a slave generate a referral message that gives the address of the master server. It is the client's responsibility to chase the referral and retry the update. OpenLDAP has no built-in way of distributing queries across replicated servers; you must use an IP sprayer/fanout program, such as balance.

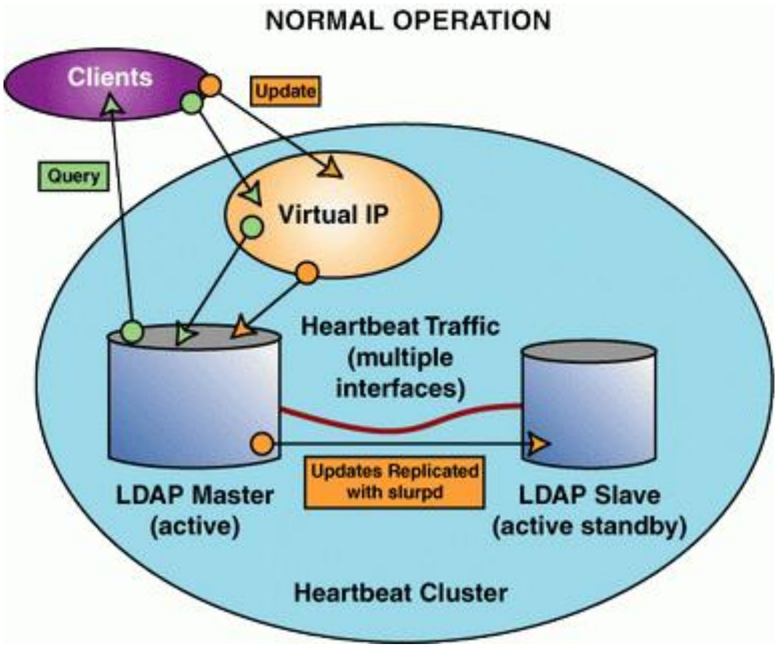
To achieve our reliability goals we cluster together a pair of servers. We could use shared storage between these servers and maintain one copy of the data. For simplicity, however, we choose to use a shared-nothing implementation, where each server has its own storage. LDAP databases typically are small, and their update frequency is low. (Hint: if your LDAP dataset *is* large, consider dividing the namespace into smaller pieces with referrals.) The shared-nothing setup does require some care when restarting a failed node: any new changes must be added to the database on the failed node before restart. We'll show an example of that situation later.

Cluster Software and Configuration

To start, let's clear up a minor confusion. Most HA (high availability) clusters have a system keep-alive function called the heartbeat. A heartbeat is used to monitor the health of the nodes in the cluster. The Linux-HA (www.linux-ha.org) group provides open-source clustering software named, aptly enough, Heartbeat. This naming situation can lead to some confusion. (Well, it confuses *us* sometimes.) In this paper, we refer to the Linux-HA package as Heartbeat and the general concept as heartbeat. Clear, yes?

The Linux-HA Project began in 1998 as an outgrowth of the Linux-HA HOWTO, written by Harald Milz. The project is currently led by Alan Robertson and has many other contributors. Version 0.4.9 of Heartbeat was released in early 2001. Heartbeat monitors node health through communication media, usually serial and Ethernet links. It is a good idea to have multiple redundant media. Each node runs a dæmon process called heartbeat. The master dæmon forks child read and write processes to each heartbeat media, along with a status process. When a node death is detected, Heartbeat runs shell scripts to start or stop services on the secondary node. By design, these scripts use the same syntax as the system init scripts (normally found in /etc/init.d). Default scripts are furnished for filesystem, web server and virtual IP failovers.

Starting with two identical LDAP servers, several configurations can be used. First we could do a “cold standby”, where the master node would have a virtual IP and a running server. The secondary node would be sitting idle. When the master node fails, the server instance and IP would move to the cold node. This is a simple setup to implement, but data synchronization between the master and secondary servers could be a problem. To solve that, we can instead configure the cluster with live servers on both nodes. This way, the master node runs the master LDAP server, and the secondary node runs a slave instance. Updates to the master are immediately pushed to the slave via slurpd (Figure 2).


NORMAL OPERATION
Clients
Update
Query
Virtual IP
Heartbeat Traffic (multiple interfaces)
Updates Replicated with slurpd
LDAP Master (active)
LDAP Slave (active standby)
Heartbeat Cluster
Figure 2. slurpd pushes updates from the LDAP master to the LDAP slave.

Failure of the master node leaves our secondary node available to respond to queries, but now we cannot update. To accommodate updates, on a failover we'll restart the secondary server and promote it to the master server position (Figure 3).

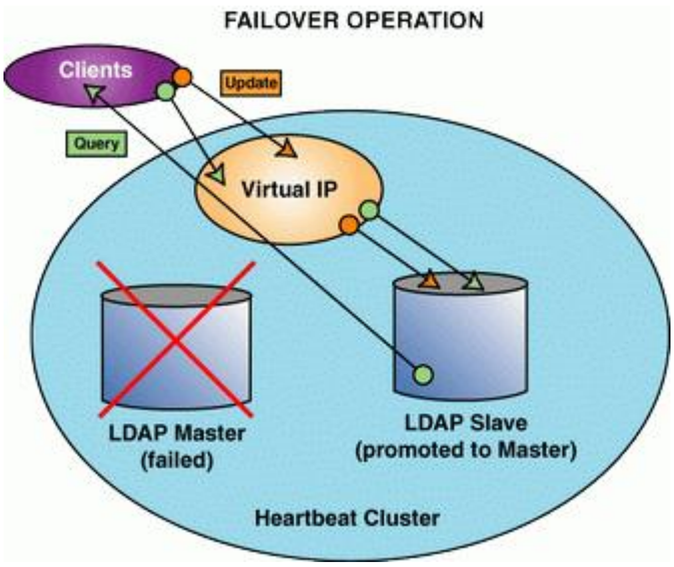


Figure 3. The LDAP slave restarts as the master.

This second configuration gives us full LDAP services, but adds one gotcha. If updates are made to the secondary server we'll have to fix the primary one before allowing it to restart. Heartbeat supports a nice failback option that bars a failed node from re-acquiring resources after a failover, an option that would be preferable. So, we'll show a restart by hand. Our sample configuration uses the Heartbeat-supplied virtual IP facility.

If heavy query loads need to be supported, the virtual IP could be replaced with an IP sprayer that distributes queries to both master and slave servers. In this case, update requests made to the slave would result in a referral. Follow-up on referrals is not automatic, so the functionality must be built into the client application. The master and slave nodes are identically configured except for the replication directives [see the Sidebar on the *LJ* FTP site, ftp.linuxjournal.com/pub/lj/listings/issue104/5505.tgz]. The master configuration file indicates the location of the replication log file and contains a listing of the slave servers, which are replication targets with credential information:

```
replica host=slave5:389
binddn="cn=Manager,dc=lcc,dc=ibm,dc=com";
bindmethod=simple credentials=secret
```

The slave configuration file does not indicate the master server. Rather it lists the credentials needed for replication:

```
updatedn "cn=Manager,dc=lcc,dc=ibm,dc=com"
```

General Heartbeat Preparation

Several good examples of basic Heartbeat configuration are available (see Resources). Here are the relevant bits from our configuration. Our configuration is quite simple, so there aren't many bits. By default, all configuration files are kept in /etc/ha.d/.

The ha.cf file that contains global definitions for the cluster is as follows:

```
# Timeout intervals
keepalive 2
# keepalive could be set to 1 second here
deadtime 10
initdead 120
# serial communications
serial  /dev/ttyS0
baud    19200
# Ethernet communications
udpport 694
udp     eth1
# and finally, our node ids
# node  nodename (must match uname -n)
node    slave5
node    slave6
```

The file haresources is where the failover is configured. The interesting stuff is at the bottom of the file:

```
slave6 192.168.10.51 slapd
```

With this line, we have indicated three things. First, the primary owner of the resource is the node slave6 (this name *must* match the output of **uname -n** of the machine you intend to be the primary machine). Second, our service address, the virtual IP, is 192.168.10.51 (this example was done on a private lab network, thus the 192.168 address). Finally, we indicated that the service script is called slapd. Therefore, Hearbeat will look for scripts in /etc/ha.d/resource.d and /etc/init.d.

The Service Script

For the simple cold standby case, we could use the standard /etc/init.d/slapd script without modification. We'd like to do some special things, however, so we created our own slapd script, which is stored in /etc/ha.d/resource.d/. [The script itself is available from the *Linux Journal* FTP site at ftp.linuxjournal.com/pub/lj/listings/issue104/5505.tgz.] Heartbeat places this directory first in its search path, so we do not have to worry about the /etc/init.d/slapd script being run instead. But, you should check to be certain slapd is no longer started on boot (remove any S*slapd files from your /etc/rc.d tree).

In the startup script, we indicate two different startup configuration files for the slapd server, allowing us to start the machine as either master or slave. When the script runs, it first stops any instances of slapd currently running. Then, if both the primary and secondary nodes are up, we start slapd as master if we're running on the primary, or we start slapd as slave if we're running on the secondary. If only one node is up, no matter which node we're running on, we start slapd as master. We do this because the virtual IP is tied to the slapd master.

To accomplish this, we must know which node is executing the script. If we are the primary node, we also need to know the state of the secondary node. The important information is in the “start” branch of the script. Because we have indicated a primary node in the Heartbeat configuration, we know when the test_start() function runs, it is running on the Heartbeat primary. (Because Heartbeat uses /etc/init.d/ scripts, all scripts are called with the argument **start|stop|restart**.)

When calling a script, Heartbeat sets many environment variables. The one we're interested in is HA_CURHOST, which has the value slave6. We can use the HA_CURHOST value to tell us when we are executing on the primary node, slave6, and when we are in a failover (HA_CURHOST would be slave5).

Now we need to know the state of the other node, so we ask Heartbeat. We'll use the provided api_test.c file and create a simple client to ask about node status. (The api_test.c file does a lot more with the client; we simply removed the bits we didn't need and added one output statement.) After compiling, we installed it in /etc/ha.d/resource.d/ and named it other_state.

Startup Script Testing

We can now start Heartbeat on both servers. The Heartbeat documentation includes some information about testing the basic setup, so we won't repeat that. With two heartbeat media, Ethernet and serial, connected, you should see six heartbeat processes running. To verify failover, we did several tests. To provide a client for testing, we created a simple KDE application that queries the servers and displays the state of the connection. A real client would query only the virtual IP in this instance, but we query all three IPs for illustration purposes. We send 10,000 queries per hour for this test (Figure 4).

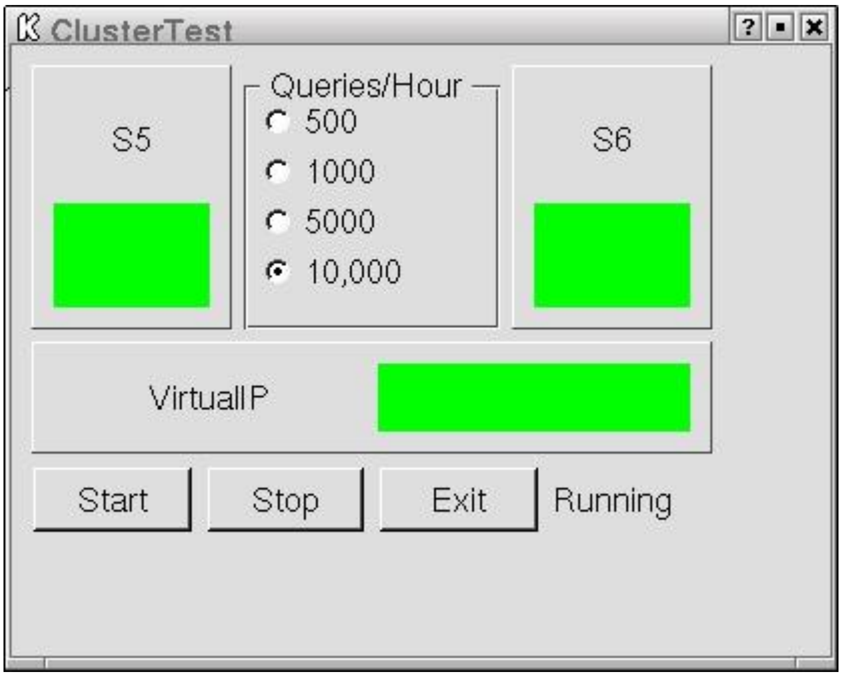


Figure 4. S6 (the master) and S5 (the standby) both running.

S6 is our master LDAP server, and Figure 4 shows that S5 is the active standby. The Virtual IP is the lower box. In the normal state, both S5 and S6 show green, indicating successful queries.

We start the test by stopping the heartbeat process on the master node. In this case the slave machine acquires the resources after the ten-second node timeout occurs, as shown in the log excerpt. The takeover includes an additional delay of two seconds inside the startup script (Figure 5).

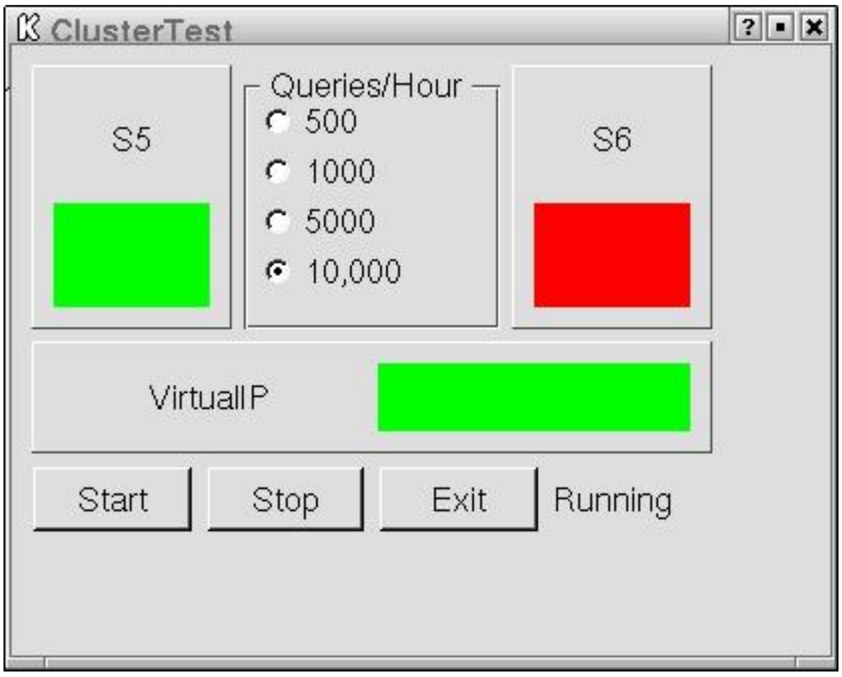


Figure 5. With the primary down, the secondary takes over the virtual IP address.

When the primary goes down, the virtual IP is serviced by the secondary, as shown in Figure 5. S5 and the virtual IP show green; server S6 is unavailable, and the indicator is red. After restarting the cluster, we created a failure by removing power from the primary node. Again the resources were acquired by the secondary node after the ten-second timeout expired.

Finally, we simulated a complete failure of the interconnects between the two nodes by unplugging both the serial and Ethernet interfaces. This loss of internode communication resulted in both machines attempting to act as the primary node. This condition is known as “split-brain”. The default behavior for Heartbeat in this case shows why it requires multiple interconnected media using separate media. In a shared-storage setup, the storage interconnect also can be used as heartbeat media, which decreases the chance of a split-brain.

This problem should be considered when choosing timeout values. If the timeout is too short, a heavily loaded system may falsely trigger a takeover, resulting in an apparent split-brain shutdown. See the Linux-HA FAQ document for more information on this.

Recovery after a Failover

If updates have been made to the LDAP namespace while the master LDAP server is down, the LDAP databases must be resynchronized prior to restarting the master server. There are two methods for doing this. If a service interruption is possible, the databases can be hand-copied after the LDAP server has been stopped. (Data files are kept by default in /usr/local/var.)

You also can use OpenLDAP replication to restore the database without the service interruption. First, start the LDAP server on the former master node as a slave. Then start slurpd on the current master. Changes received while the former master was out of service are pushed from the new master. Finally, stop the slave LDAP server on the former master node, and start Heartbeat. This results in a failback to the original configuration.

Conclusions

This article outlines a simple example of using open-source software to create some highly available basic network services. Network services including LDAP seldom require huge servers. The additional reliability provided by clustering and the duplication of servers and data files can increase overall service availability. The system worked under all tests, with a failover of less than 15 seconds in all cases. Given a good understanding of system loads and utilization, failover time could be reduced below this threshold.

Acknowledgements

Thanks to Alan Robertson, IBM Linux Technology Center, for his helpful comments and review.

Disclaimer

Resources

Jay D. Allen works by day on the leading edge of IT as a software engineer at the IBM Linux for Service Providers Lab (LSPL), working with Linux on Intel and RS/6000. By night he works on the trailing edge of IT, mostly with DEC PDP-11s and other antiques. He can be reached at allen5@us.ibm.com.

Cliff White is a member of the technical staff working for the Open Source Development Lab (www.osdl.org). He has worked with various flavors of UNIX and Linux since 1989. He authenticates himself each morning, just to be sure. He can be reached at cliffw@osdl.org.

Process Accounting

Keith Gilbertson

Issue #104, December 2002

One notch in your security belt, maybe for tracking gaming time, here's some basic how-tos.

While on site at a Fortune 500 corporation recently, I overheard a tech support person whispering excitedly to a project manager, “Don't play any games on your PC! The corporate auditors have a way to find out exactly what programs you use and for how long!”

After loudly assuring the techie that he was all business and didn't intend to play games anyway, the manager smiled. Then in a much quieter tone he said he needn't be concerned; he was using Linux and not Windows, unlike most of the company.

If the tech's tale is true, the manager may indeed have reason for concern. Although the rumoured auditing application at this particular company was developed for Windows, the Linux kernel has a built-in process accounting facility. It allows system administrators to collect detailed information in a log file each time a program is executed on a Linux system. With this capability, our mythical corporate auditor could, in fact, collect information about who has been playing games on a Linux computer and for how long.

Although a company's interest in knowing which employees have been indulging in *Solitaire* on company equipment is of questionable merit, there are good reasons to use process accounting (PA). In this article, I discuss some situations where process accounting is useful, explain where to obtain and how to use the standard process accounting commands, and then demonstrate how to use the process accounting structure and system call in C programs.

Preliminaries

I assume that your system has process accounting support compiled into the kernel. I make this assumption because the kernels on all of the Linux systems I have had access to are configured to allow process accounting, but your distribution may be different. If you compile and run the first code listing in this article as root with no command-line arguments but receive an error message, it is likely that process accounting support is not included in your kernel. You'll need to compile a new kernel and answer yes to CONFIG_BSD_PROCESS_ACCOUNTING, which is the BSD Process Accounting item in the General Setup menu. Recompiling your kernel is beyond the scope of this article, but instructions can be found at the Linux Documentation Project (www.tldp.org/HOWTO/Kernel-HOWTO.html).

On busy systems, keep in mind that turning on process accounting requires significant disk space. On my Pentium III system with Red Hat 7.2, each time a program is executed, 64 bytes of data are written to the process accounting log file. While researching this article and running the process accounting utilities on a test machine with low disk space, I discovered a monitor process that executes every second. The drive on that machine filled up quickly. Some server's dæmons will initiate a separate process for each incoming connection. On a production server that executes nearly 25,000 processes per hour, approximately 1.1GB of process accounting data is generated each month. Utilities, such as the accttrim and handleacct.sh script listed in Table 1, are available to truncate, back up and compress log files at regular intervals. If you plan on doing process accounting on a busy system, it will be important for you to learn about and use these utilities.

Finally, know that you must have root privileges on your Linux system to enable or disable process accounting, whether using the standard commands or creating your own.

Uses of Process Accounting

One of the earliest uses of process accounting was to calculate the CPU time absorbed by users at computer installations and then bill users accordingly. With the greater abundance and relatively low expense of today's computing resources, this application has fallen by the wayside. If the distributed computing model catches on, however, this application could again become important.

System administrators may wish to use data collected from the PA facilities to monitor which programs are most accessed by users, and then optimize the system configuration for these types of programs. For example, part of the data collected by the PA facilities includes the number of bytes that are input and output by the program and the CPU usage. A system that runs a high percentage of I/O-intensive applications may need to be optimized in ways that a system running a high percentage of CPU-bound applications not.

At some point an administrator might be required to evaluate two products with similar functionality. Let's imagine that before making a selection, the administrator wishes to see which fish forecasting product the people are actually using. To do this, process accounting can be turned on for a week to record the names of all the commands executed in a log file. The administrator can then parse the log file to find out which command was run more often.

The most typical application of process accounting is as a supplement to system security measures. In the case of a break-in on a company server, the log files created by the process accounting facility are useful for collecting forensic evidence. A careful look at the programs an attacker has used on the compromised system can provide useful information about the damage done, as well as the intruder's methods and possible motivations. Evidence collected from the process accounting logs also may be helpful in court. I know of one criminal case in which this data, when uncontested by the defendant, led to a misdemeanor conviction.

Standard Process Accounting Commands

Even if process accounting facilities have been compiled into your kernel, you might not have the user commands for process accounting installed on your system. If this is the case, and you're looking to get started quickly, first try finding the process accounting commands for your specific Linux distribution.

The package for your distribution likely is configured to place log files in the appropriate location for your system's setup, making installation much simpler. On my Red Hat 7.2 distribution CDs, I found the ps-acct-6.3.2-9.i386.rpm on the second disk, in the directory. If you use the gnorpm graphical install tool, the package will appear in the Packages/Applications/System hierarchy. On a Debian system, install the acct package.

If you're installing from source, two versions of the utilities are available. One version, under the BSD license, is available at www.ibiblio.org/pub/Linux/system/admin/accounts. The filename will be similar to acct-1.3.73.tar.gz, with small differences depending on the version number. In order to get these utilities to compile on my system, I had to edit the lastcomm.c file and comment out the prototype for the strcpy function.

There is also a process accounting utilities set written by Noel Cragg and licensed under the GNU GPL. It's available at www.gnu.org/directory/System_administration/Monitoring/acct.html.

The exact process accounting commands installed on your system will vary depending on the particular package you've chosen. Table 1 shows a list of the commands you could encounter and the purpose of each.

Table 1. Process Accounting Commands

Installation of the GNU Accounting Utilities

Let's take a quick look at how to install the GNU Accounting Utilities on a system. Use the following commands:

```
tar zxvf acct_6.3.5.orig.tar.gz
cd acct-6.3.5
./configure
make
su
make install
```

A few basic process accounting commands have now been installed on your system. You're now ready to turn on the accounting and start using the commands.

Using the Utilities

In this brief introduction to using the process accounting commands, I look at two commands, accton and lastcomm. I've chosen these two commands because they are standard on all process accounting versions.

The accton command switches process accounting on or off. If a filename is specified on the command line, that filename will be used to log the process accounting information. If no argument is specified, process accounting will be switched off.

To start the process accounting facilities on your system, **su** to become root. Make sure that the log file exists by performing a **touch** on the desired location. Example:

```
touch  /var/log/pacct
```

Then type the full path to your accton program (usually /usr/sbin/accton or /sbin/accton) followed by the filename. Example:

```
/sbin/accton /var/log/pacct
```

You've just started the process accounting facilities. Note that the data actually is not added to the file when each process begins execution; it is written when a process exits. The aforementioned project manager can play the *xbill* game all day long and not have this information written to the process accounting file, as long as he never exits the program. When he goes home at night, he can choose to leave *xbill* running and minimize the window, or he can simply power off his computer without performing a proper shutdown.

Now that you've switched on the accounting, run a few normal commands as an ordinary user to get some data for the lastcomm command, which you'll use next. When you're finished, **su** to root once more, and run **/usr/sbin/accton** or **/sbin/accton** with no arguments to switch off process accounting.

The lastcomm command prints information contained in the accounting log files, with the most recent record printed first. You can use the -f command-line option to specify a filename. Typically, the process accounting log file on a system is set up so that only root can read it. This command is then executed by root, for example:

```
lastcomm -f /var/log/pacct
```

When you type in the above command, the output is similar to this:

```
id          root   stdin  0.00 secs Mon Jul 22 12:41
xauth    S  root   stdin  0.00 secs Mon Jul 22 12:41
xauth    S  keithg stdin  0.00 secs Mon Jul 22 12:41
xauth    S  keithg stdin  0.01 secs Mon Jul 22 12:41
bubbles   X keithg ??     0.01 secs Mon Jul 22 12:33
ls          keithg ??     0.01 secs Mon Jul 22 12:26
bash      X keithg ??     0.03 secs Mon Jul 22 08:25
```

lastcomm displays the command name, options, user name, terminal and exit time for each command. A particular command, user or terminal also can be specified on the command line. For example, if you want to find instances only of when the su program was started, you can type:

```
lastcomm -f /var/log/pacct --command su
```

Now you'll see output like this:

```
su      root     ??      0.01 secs Mon Jul 22 10:52
su      keithg   stdout  0.05 secs Mon Jul 22 09:32
su      keithg   stdout  0.00 secs Mon Jul 22 09:17
su      root     ??      0.00 secs Mon Jul 22 03:29
su      keithg   tty1    0.00 secs Sun Jul 21 19:49
```

Notice that on each line, the command listed in the left column is now su. For more details about these commands and the other programs in the table, see the respective man pages.

Programming Details

The acct structure for collecting process accounting details is documented in the header files /usr/include/linux/acct.h and /usr/include/sys/acct.h. Table 2 displays the members available in the acct struct and a brief description of each member.

Table 2. Members in the acct struct

As you can see from the table, a lot of information is packed into the 64-byte accounting record. If you feel you need more information than is available with standard process accounting, consult the book by Mann and Mitchell listed in Resources at the end of this article.

Example Programs

Listing 1. Enabling and Disabling Accounting to a File

Listing 1 is a simple demonstration of the use of the acct system call. The acct call takes one argument, the name of the file to which process accounting information is appended. If the argument is NULL, process accounting will be turned off. In addition, the file already must exist when the system call is made, or the call will fail and an error will be returned.

If a program running with the ID of an ordinary user makes a call to acct, the call also will fail and return an error. Programs that attempt to switch process accounting on and off must have root privileges to succeed.

The code in Listing 1 is similar to a typical implementation of the accton command, but there are two main differences. The first is that this code will report its actions in messages to standard output. The second is that if the file specified on the command line does not exist, it will be created.

The file includes the <unistd.h> header file. All programs that make use of the acct call should include this file. The program checks to see if argc is equal to one, meaning no arguments were passed on the command line. If this is so, the program attempts to turn off process accounting by calling acct with a NULL argument.

If the command is run with an argument, the program will assume that the first argument is the filename. If the file does not exist, the program will attempt to create it with the creat system call. Then, the program will call acct with the filename as an argument to turn on process accounting. If an error code is returned from a system call, a message will be printed and the program will exit.

Listing 2. Parsing the Accounting File

Listing 2 demonstrates how to read records from the log file into an acct structure in memory so that the information can be printed out or operated upon. This program includes the <sys/acct.h> header file. All programs that need to work with the acct structure should include this file. Local variables in the main function include a file descriptor, a variable to hold the number of bytes read from the file and an acct struct.

The user of the program must specify a filename on the command line. The program attempts to open this file for read-only access. If the open was successful, the program will read() a record from the file directly into the local acct structure, a. Due to space constraints for the article, I've made the assumption that a read() always will return exactly the number of bytes requested, until the end of file is reached. The program continues to read and print the command name from the records until a zero is returned from the read() call, signalling the end of file condition.

The Listings in this article are intended to be simple introductions to the system accounting structures. Robust programs would create a buffer to read multiple accounting records at once, and they would check for issues such as fewer bytes read from the file than were requested. To see examples of robust programs, look at the source for the process accounting utilities that you've installed.

Conclusion

You now have enough information to enable process accounting and use the standard commands to retrieve information about programs executed on a Linux system. If you're so inclined, you also can learn to make custom tools that parse the process accounting log files.

If you're using process accounting for system security, keep in mind that it is not by any means a comprehensive solution, but only one small tool. In fact, as Mann and Mitchell point out, you should be careful about trusting the information in the process accounting log files; the logs may have been modified by a technically savvy attacker.

With a basic understanding of the process accounting tools in Linux and some experimentation, you can set up these utilities on your own computer. If you're fortunate enough to have root access to the systems at work, you'll also be prepared to remove all traces of the *Sokoban* game from the accounting log files—in case that evil corporate auditor really does show up in your department one day.

Resources

Keith Gilbertson (keithg@kellnet.com) is a graduate of Bowling Green State University in Ohio. He works as a programmer analyst on the wireless and Linux development teams at the May Company's Data Center near Lake Erie. On the lake, the fish fear no penguins.

Archive Index Issue Table of Contents

Advanced search

OpenLDAP Everywhere

Craig Swanson

Matt Lung

Issue #104, December 2002

Step-by-step instructions for sharing e-mail directories, having a unified login and sharing files in a mixed environment.

The purpose of this article is to demonstrate the use of OpenLDAP as the core directory service for a heterogeneous environment. The LDAP server provides a shared e-mail directory, a unified login for Linux and Windows users, automount of home directories and file sharing for both Linux and Windows clients.

Midwest Tool & Die has been using OpenLDAP for three years, and the performance has been flawless. We have experienced 100% uptime for the directory. The company saw the first big benefit from sharing e-mail contacts in the directory. Now, we have unified logon from any networked computer. Our computer users can access the same file storage through Windows/Samba or through Linux/NFS/automount. The result is seamless access to network services.

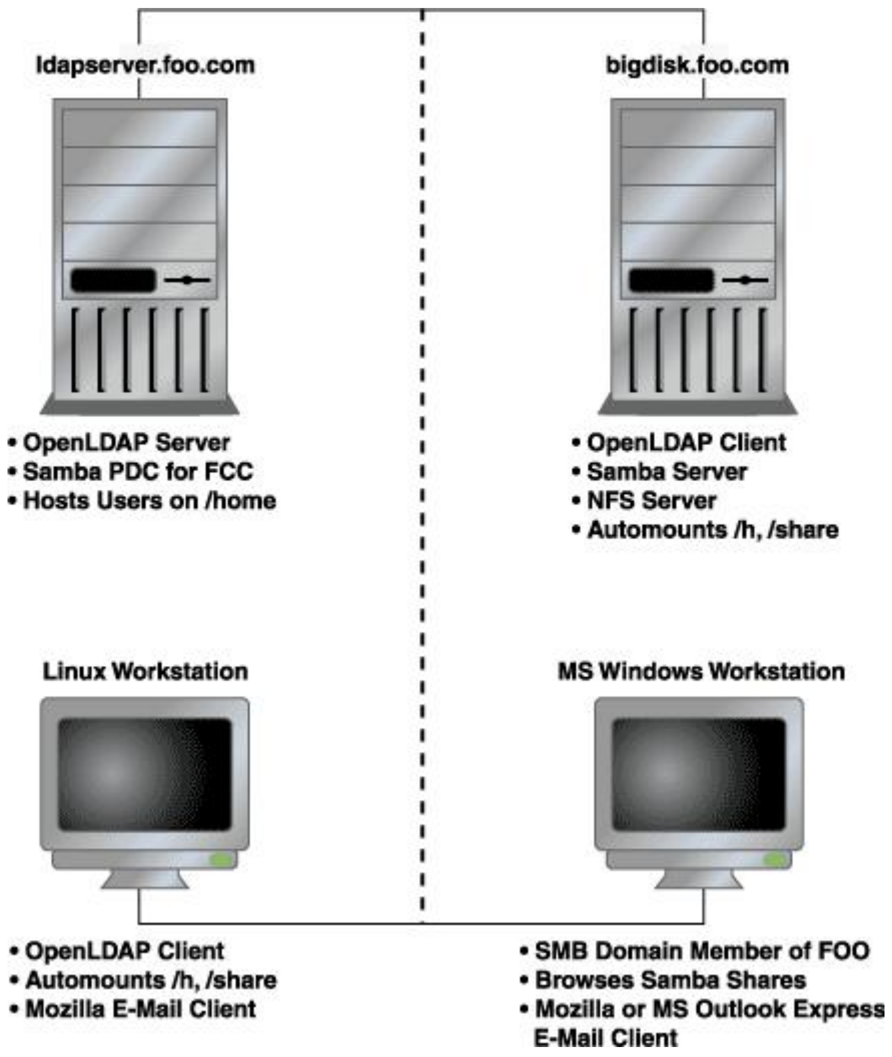


Figure 1. OpenLDAP Mixed Environment

A simple mixed environment used in the examples in this article is shown in Figure 1. The configuration discussed in this article does not document the use of SSL. The ldapsync.pl program it uses may expose your LDAP manager password. As a result, Windows clients may cache user passwords, thereby creating a new risk to Linux security. Review your security needs with caution and prudence, and attempt this configuration at your own risk. Neither the authors, nor our employer, Midwest Tool & Die, takes any responsibility for your security.

LDAP Server Installation and Configuration

The LDAP server we discuss was installed using RPM binary packages and uses openldap-2.0.11-8 on Red Hat 7.1. You also need to have the auth_ldap and nss_ldap packages. This article assumes a domain name of foo.com.

To use the most recent source, follow the instructions at www.openldap.org/doc/admin/quickstart.html to download and install OpenLDAP. Edit the OpenLDAP server configuration file, /etc/openldap/slapd.conf as follows:

```
# Schemas to use
include  /etc/openldap/schema/core.schema
include  /etc/openldap/schema/cosine.schema
include  /etc/openldap/schema/inetorgperson.schema
include  /etc/openldap/schema/nis.schema
include  /etc/openldap/schema/redhat/
rfc822-MailMember.schema
include  /etc/openldap/schema/redhat/autofs.schema
include  /etc/openldap/schema/redhat/
kerberosobject.schema
database       ldbm
suffix         "dc=foo,dc=com"
rootdn         "cn=Manager, dc=foo,dc=com"
rootpw         {crypt}sadtCr0CILzv2
directory      /var/lib/ldap
index   default                       eq
index   objectClass,uid,uidNumber,gidNumber eq
index   cn,mail,surname,givenname           eq,sub
# Access Control (See openldap v.2.0 Admin Guide)
access to attr=userPassword
   by self         write
   by anonymous    auth
   by dn="cn=manager,dc=foo,dc=com"       write
   by *    compare
access to *
   by self write
   by dn="cn=manager,dc=foo,dc=com"       write
   by * read
```

The LDAP schemas define object classes and attributes that make up the directory entries. With the edits above, the hard work of defining schemas to fit our uses has been done. The schemas that we need, listed in the first section of slapd.conf, already have been defined and packaged with the RPM installation.

If you find that you need to add an objectClass or an attribute for your directory, see the OpenLDAP admin guide at www.openldap.org/doc/admin20/schema.html. We'll use the default database type ldbm, and our example uses the LDAP domain component. Therefore, foo.com becomes dc=foo,dc=com. In addition, the manager has full write access to LDAP entries.

The Red Hat 7.3 Reference Guide suggests using **crypt** to protect the manager's password:

```
perl -e "print crypt('passwd',
'salt_string',);"
```

In the previous Perl line, replace *salt_string* with a two-character salt, and *passwd* with the plain-text version of the password. Paste the resulting encrypted password into slapd.conf as shown above.

The index lines enhance performance for attributes that are often queried. Access control restricts access to the userPassword entry, but the user and manager may modify the entry. For all other entries, the manager has write access, and everyone else is granted read access.

Create the Directory Structure

LDAP can be seen as a tree, with foo.com at the trunk. Branches are created as organizational units (ou), as shown in Figure 2.

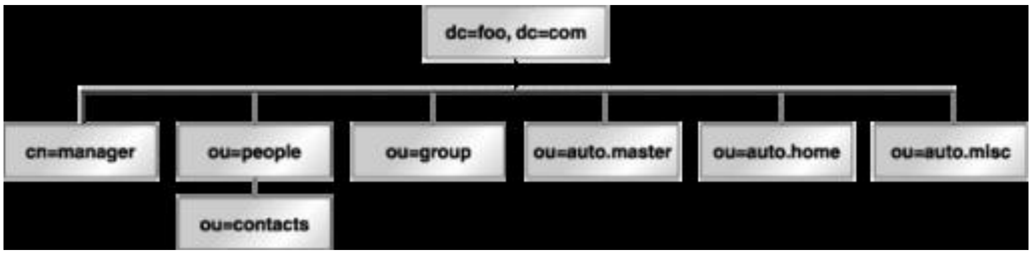


Figure 2. Organizational units are branches on the LDAP tree.

Each entry in the directory is uniquely identified with a distinguished name (dn). The dn for the LDAP manager looks like dn: cn=manager, dc=foo, dc=com.

The ou provides a method for grouping entries, as shown in Table 1.

Table 1. ou Method for Grouping Entries

We create the individual entries in LDIF (LDAP Interchange Format) and save them to top.ldif:

```
dn: dc=foo, dc=com
objectclass: dcObject
objectclass: organization
o: Foo Company
dc: foo
dn: cn=manager, dc=foo, dc=com
objectclass: organizationalRole
cn: manager
dn: ou=people, dc=foo, dc=com
ou: people
objectclass: organizationalUnit
objectclass: domainRelatedObject
associatedDomain: foo.com
dn: ou=contacts, ou=people, dc=foo, dc=com
ou: contacts
ou: people
objectclass: organizationalUnit
objectclass: domainRelatedObject
associatedDomain: foo.com
dn: ou=group, dc=foo, dc=com
ou: group
objectclass: organizationalUnit
objectclass: domainRelatedObject
```

Add the top-level entries to the directory with **ldapadd**:

```
ldapadd -x -D 'cn=manager,dc=foo,dc=com' -W \
-f top.ldif
```

Then, test your work with **ldapsearch** to retrieve all entries:

```
ldapsearch -x -b 'dc=foo,dc=com'
```

Share E-Mail Contacts

At this point, we have enough structure in LDAP to put it to real use. We'll start by sharing our e-mail contacts, which also should be in LDIF.

To simplify the process, you may be able to export your e-mail address book in LDIF. For example, in Mozilla 1.0, you can export in LDIF from the Tools menu on the address book window. Microsoft Outlook Express also allows exporting the address book in LDIF. You will need to process the resulting file so it looks like our contacts example below; I suggest using Perl for the task.

Contacts are uniquely identified by their e-mail addresses. Here is the dn for a sample contact:

```
dn: uid=someone@somewhere.com,ou=contacts,
    ou=people, dc=foo,dc=com
```

With all of the attributes, the full entry for a contact looks like:

```
dn: uid=someone@somewhere.com,ou=contacts,
    ou=people, dc=foo,dc=com
cn: Someone Youknow
mail:
uid:
givenname: Someone
sn: Youknow
objectclass: person
objectClass: top
objectClass: inetOrgPerson
```

Separate each contact entry with a blank line, and save it to a file called contacts.ldif. Then you can add the contacts to the directory with **ldapadd**:

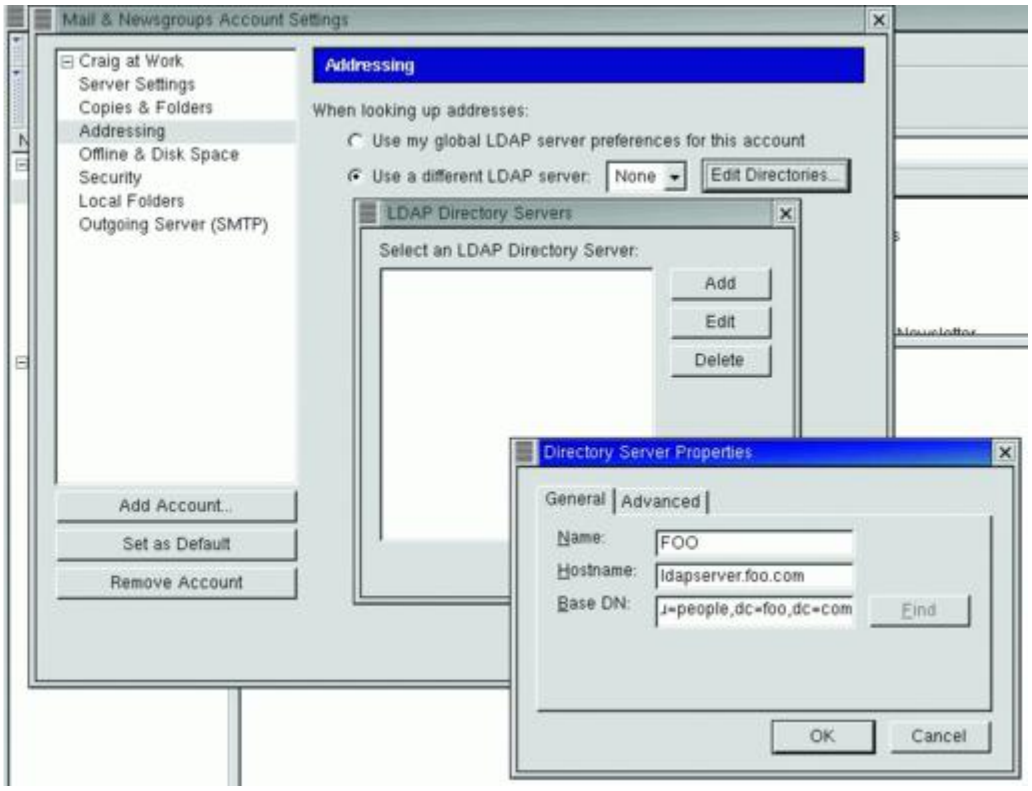
```
ldapadd -x -D 'cn=manager,dc=foo,dc=com' -W \
-f contacts.ldif
```

Once again, test your work with an **ldapsearch** that retrieves all entries:

```
ldapsearch -x -b 'dc=foo,dc=com'
```

Configure E-Mail Clients

Now it's time to configure Mozilla to use the new LDAP server (see Figure 3).



Name: FOO
Hostname: ldapserver.foo.com
Base DN: u=people,dc=foo,dc=com

Figure 3. Directory Server Properties Dialog Box in Mozilla

From the Edit menu in the Mozilla Mail and News window, select Mail & Newsgroup Account Setting. In the Addressing tab, select Use a different LDAP server, then select Edit Directories and then Add. Fill in the Directory Server Properties dialog with:

```
Name: FOO
Server: ldapserver.foo.com
base DN: ou=people,dc=foo,dc=com
```

Next, tell Mozilla to look up addresses in your directory. Under Addressing in the Mail and Newsgroups preferences, select Address Autocompletion and fill in FOO for Directory Server.

Test your settings by composing a message to one of your contacts in your LDAP directory. The address should autocomplete as you type. Another test is to search the LDAP directory from within the Mozilla Mail Address Book. A search for Name or E-mail that contains * should return all of the contact entries. Similarly, you can also configure Microsoft Outlook Express to use the LDAP directory.

Unified Linux Login with LDAP

By storing user account information in LDAP, you can use the same user name and password at any Linux console. To start, you must decide which user names should be entered in LDAP. Here is our user scheme for UID/GIDs:

- System accounts: UID < 500
- Real people in LDAP: 499 < UID < 10,000
- Local users, groups (not in LDAP) > 10,000

This user scheme allows for 9,500 LDAP user and group entries, while allowing local per-system users and groups that do not interfere with LDAP UID/GIDs.

Create Local Computer User Entries

An entry for a local computer user is identified by the login name as “uid”. Local computer users are members of ou=people: dn: uid=gomerp,ou=people,dc=foo,dc=com.

The full entry contains the attributes needed to control account access:

```
dn: uid=gomerp,ou=people,dc=foo,dc=com
uid: gomerp
cn: Gomer Pyle
givenname: Gomer
sn: Pyle
mail:
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: account
objectClass: posixAccount
objectClass: top
objectClass: kerberosSecurityObject
objectClass: shadowAccount
userPassword: useradd_ldap_flag
shadowLastChange: 11547
shadowMax: 99999
shadowFlag: 0
krbname:
loginShell: /bin/bash
uidNumber: 531
gidNumber: 531
homeDirectory: /h/gomerp
gecos: Gomer Pyle
```

To make this easier, OpenLDAP ships with migration utilities that can extract the user account information; see /usr/share/openldap/migration. The first thing you need to do is edit migrate_common.ph:

```
# Default DNS domain
$DEFAULT_MAIL_DOMAIN = "foo.com";
# Default base
$DEFAULT_BASE = "dc=foo,dc=com";
# turn this on to support more general object classes
# such as person.
$EXTENDED_SCHEMA = 1;
```

Then, extract the user account information:

```
/usr/share/openldap/migration/migrate_passwd.pl \
/etc/passwod >people.ldif
```

Once this is done, review the resulting LDIF file. You should remove entries for system accounts such as root and for local system users that do not need to appear in LDAP. Finally, add the user entries to LDAP:

```
ldapadd -x -D 'cn=manager,dc=foo,dc=com' -W \
-f people.ldif
```

As always, test your work with an **ldapsearch** that retrieves all entries:

```
ldapsearch -x -b "dc=foo,dc=com"
"(objectclass=*)"
```

Because the computer users belong to ou=people, you may now look up their e-mail addresses within your mail client.

Create Group Entries

You need to make a group entry for each group that is shared between multiple Linux computers. Each user also needs a group entry for the user private group. A group entry is identified by “cn”, and each group belongs to ou=group, for example:

```
dn: cn=gomerp,ou=group,dc=foo,dc=com
```

A user private group would look like this:

```
dn: cn=gomerp,ou=group,dc=foo,dc=com
objectClass: posixGroup
objectClass: top
cn: gomerp
userPassword: {crypt}x
gidNumber: 531
```

While a shared group would look like:

```
dn: cn=web_dev,ou=group,dc=foo,dc=com
objectClass: posixGroup
objectClass: top
cn: web_dev
gidNumber: 502
memberUid: gomerp
memberUid: goober
memberUid: barneyf
```

After creating the group entry, extract the group information:

```
/usr/share/openldap/migration/migrate_passwd.pl \
/etc/group >group.ldif
```

Review the resulting LDIF file, removing entries for system groups and for local system users that do not need to appear in LDAP. Then, add the group entries to LDAP:

```
ldapadd -x -D 'cn=manager,dc=foo,dc=com' -W \
-f group.ldif
```

Test your work with an **ldapsearch** that retrieves all group entries:

```
ldapsearch -x -b 'dc=foo,cd=com'
```

Configure Automount to Share Home Directories (and NFS Shares)

With unified login, users have a single home directory shared via NFS. To keep things simple, we host our home directories from ldapserver.foo.com and share /home via NFS. NFS is outside the scope of this article, but here is a line from /etc/exports that works.

```
/home *.foo.com(rw)
```

Linux LDAP clients mount the user's home directory at login, using automount and NFS. The LDAP use of automount is a replacement for NIS (Network Information Service) automount maps. Replace the automount maps for auto.master, auto.home and auto.misc.

We also create a new organizational unit for auto.master:

```
dn: ou=auto.master,dc=foo,dc=com
objectClass: top
objectClass: automountMap
ou: auto.master
```

An auto.master entry is identified by “cn”. The automountInformation attribute instructs automount to look for the map in LDAP:

```
dn: cn=/h, ou=auto.master,dc=foo,dc=com
objectClass: automount
automountInformation: ldap:ou=auto.home,
    dc=foo,dc=com
cn: /h
```

While we're at it, let's create an auto.master entry for other NFS shared directories:

```
dn: cn=/share, ou=auto.master,dc=foo,dc=com
objectClass: automount
automountInformation: ldap:ou=auto.misc,
    dc=foo,dc=com
cn: /share
```

Create the automount entries in LDIF format and save as auto.master.ldif:

```
dn: ou=auto.master,dc=foo,dc=com
objectClass: top
objectClass: automountMap
ou: auto.master
dn: cn=/h, ou=auto.master,dc=foo,dc=com
objectClass: automount
automountInformation: ldap:ou=auto.home,
    dc=foo,dc=com
cn: /h
dn: cn=/share, ou=auto.master,dc=foo,dc=com
objectClass: automount
automountInformation: ldap:ou=auto.misc,
    dc=foo,dc=com
cn: /share
```

Add the auto.master entries to LDAP:

```
ldapadd -x -D 'cn=manager,dc=foo,dc=com' -W \
-f auto.master.ldif
```

Next, we create a new organizational unit for auto.home, ou=auto.home. A home directory entry is identified by "cn":

```
dn: cn=gomerp,ou=auto.home,dc=foo,dc=com
```

Create auto.home entries for each user in LDIF format and save as auto.home.ldif:

```
dn: ou=auto.home,dc=foo,dc=com
objectClass: top
objectClass: automountMap
ou: auto.home
dn: cn=gomerp,ou=auto.home,dc=foo,dc=com
objectClass: automount
automountInformation:
    ldapserver.foo.com:/home/gomerp
cn: super3
```

Add the auto.home entries to LDAP:

```
ldapadd -x -D 'cn=manager,dc=foo,dc=com' -W \
-f auto.home.ldif
```

When automounted from a Linux LDAP client, your home directory (ldapserver.foo.com:/home/gomerp) is mounted on /h/gomerp. Other NFS shares may be entered in LDAP and automounted as they are needed. The auto.misc organizational unit holds these automount maps, which have the form ou=auto.misc.

We've already created an auto.master entry for /share, as indicated above. Now, create entries for NFS shares under auto.misc, and save them as auto.misc.ldif:

```
dn: ou=auto.misc,dc=foo,dc=com
objectClass: top
objectClass: automountMap
ou: auto.misc
dn: cn=redhat,ou=auto.misc,dc=foo,dc=com
objectClass: automount
automountInformation:
    bigdisk.foo.com:/pub/redhat
cn: redhat
dn: cn=engineering,ou=auto.misc,dc=foo,dc=com
objectClass: automount
automountInformation:
    bigdisk.foo.com:/data/engineering
cn: engineering
```

Add the auto.misc entries to LDAP:

```
ldapadd -x -D 'cn=manager,dc=foo,dc=com' -W \
-f auto.misc.ldif
```

When automounted from a Linux LDAP client, your shared directory bigdisk.foo.com:/data/engineering is mounted on /share/engineering.

Configure the Linux LDAP Client

You now need to install the authentication package, auth_ldap, and the name switch service package, nss_ldap. The Red Hat tool /usr/bin/authconfig is handy for configuring the client. Select Use LDAP®Server: ldapserver.foo.com, base DN: dc=foo,dc=com. Authconfig writes to these files: /etc/ldap.conf, /etc/openldap/ldap.conf and /etc/nsswitch.conf.

Verify that /etc/nsswitch.conf has the following entries:

```
passwd:    files ldap
shadow:    files
group:     files ldap
automount: files ldap
```

Verify that /etc/ldap.conf has these entries:

```
host ldapserver.foo.com
base dc=foo,dc=com
```

and that /etc/openldap/ldap.conf has these entries:

```
HOST ldapserver.foo.com
BASE dc=foo,dc=com
```

Final Linux Server Configuration

The LDAP server also is a client of LDAP. On the LDAP server, disable the automount of /home as /h. **nsswitch** is configured to check the files first, and then LDAP for automount information. So, we will make a dummy entry in ldapserver.foo.com:/etc/auto.master:

```
/h /etc/auto.null
```

The user's password and group entries must be removed from the password and group files on the home directory server. Create backups, then edit /etc/passwd, /etc/shadow, /etc/group and /etc/gshadow to remove the LDAP real-people entries.

To test, log in to a Linux LDAP client, using an LDAP user name. You should see the appropriate login shell and home directory for that user. To test auto.misc shares, you must access the share by name:

```
cd /share/redhat
```

Automount only mounts NFS shares as they are used, so the directory /share/redhat is not visible until it has been accessed.

Microsoft Windows Unified Login with Samba and LDAP

To have a Windows and Linux unified login, first configure a Samba Primary Domain Controller (PDC). User home directories are shared with SMB clients. The details of Samba configuration are outside the scope of this article.

Configure ldapsync.pl and Samba

User passwords may be changed from MS Windows using Samba and the Perl program ldapsync.pl, which is available from www.mami.net/univr/tng-ldap/howto/#how_to_change_password.

The ldapsync.pl script is a replacement for the /bin/passwd program called by Samba to change users' passwords, and it keeps them in sync with the Samba passwords. The ldapsync.pl script is called from Samba when changing user passwords within Windows, and it is run as root just as /bin/passwd is normally run in an unmodified Samba. The ldapsync.pl script is needed for LDAP-enabled users to function. Because the user passwords are not stored locally in /etc/passwd but in LDAP, the ldapsync.pl script binds to the LDAP directory and modifies the user's password entry in LDAP.

In simpler terms, here's how this process works:

1. User calls password-changing program from Windows.
2. User clicks OK to change password and sends data to Samba server.
3. Samba looks at its config file and knows to call ldapsync.pl to change LDAP passwords.
4. **ldapsync.pl** is executed with **-o %u** options that specify the program to run without prompting for the old password. It passes the user's name to the script as it runs (important if you don't want to change root's password without knowing it).
5. Samba passes the user's new password to ldapsync.pl without caring about what the old one was.
6. **ldapsync.pl** chats with Samba, expecting the correct responses with the new password.
7. If it passes the chat correctly, the password is encrypted by ldapsync.pl.
8. **ldapsync.pl** then binds LDAP with the correct dn of the user and does an **ldapmodify** on the user's LDAP entry, replacing the userPassword field stored in LDAP. LDAP and Samba chat for a final time, listening for success from LDAP, at which point the process ends.

To configure Samba for this, you will need the following Smb.conf entries:

```
passwd program = /etc/samba/ldapsync.pl -o %u
passwd chat = *New*password* %n\n
*Retype*new*password* %n\n *modifying*
```

When users change their passwords in Windows they are prompted for the old password, a new one and then are asked to confirm the new one. Because ldapsync.pl is called without caring about the old password, only the two new entries are examined. First of all, the * instructs it to look for anything and then a specific match. So the ***New*password*%n\n** is saying match anything, then the word New, then anything and the word password, then anything and the new password the user entered (**%n**). The ***modifying*** is saying if LDAP returns that it modified the entry, then the process was successful.

You must edit ldapsync.pl to enter the LDAP bind information:

```
$binddn = "cn=manager,dc=foo,dc=com";
$passwd = "passwd";
```

Then, limit the access of ldapsync.pl to root only (0700).

Sharing NFS Shares with Samba

Your NFS shares can be shared with Windows clients by running a Samba server on the NFS host. The Samba server must join your FOO SMB domain. Run the following command on the Samba server to join the SMB domain:

```
smbbpasswd -j [FOO] -r [PDC]
```

Maintenance

Congratulations! Your LDAP server is up and running with shared e-mail contacts, unified login and shared file storage that is accessible from any client. You probably want to write some administrative utilities to help maintain user and group accounts. Again, we recommend Perl for the task.

Credits

ldapsync.pl originally written by Jody Haynes for Samba-Tng.

Resources

Craig Swanson (craig.swanson@midwest-tool.com) is a part owner of Midwest Tool & Die and has used Linux since 1993. He designed the company network and acts as a mentor for software development and manufacturing engineering.

Matt Lung (matt.lung@midwest-tool.com) works as a Network Engineer at Midwest Tool & Die. He graduated from Purdue University in May, with a degree in Computer Engineering Technology. He configured the company's virtual private network and likes to build robots.

Playing with ptrace, Part II

Pradeep Padala

Issue #104, December 2002

In Part II of his series on ptrace, Pradeep tackles the more advanced topics of setting breakpoints and injecting code into running processes.

In Part I of this article [*LJ*, November 2002], we saw how ptrace can be used to trace system calls and change system call arguments. In this article, we investigate advanced techniques like setting breakpoints and injecting code into running programs. Debuggers use these methods to set up breakpoints and execute debugging handlers. As with Part I, all code in this article is i386 architecture-specific.

Attaching to a Running Process

In Part I, we ran the process to be traced as a child after calling ptrace(PTRACE_TRACEME, ..). If you simply wanted to see how the process is making system calls and trace the program, this would be sufficient. If you want to trace or debug a process already running, then ptrace(PTRACE_ATTACH, ..) should be used.

When a ptrace(PTRACE_ATTACH, ..) is called with the pid to be traced, it is roughly equivalent to the process calling ptrace(PTRACE_TRACEME, ..) and becoming a child of the tracing process. The traced process is sent a SIGSTOP, so we can examine and modify the process as usual. After we are done with modifications or tracing, we can let the traced process continue on its own by calling ptrace(PTRACE_DETACH, ..).

The following is the code for a small example tracing program:

```
int main()
{   int i;
    for(i = 0;i < 10; ++i) {
        printf("My counter: %d\n", i);
        sleep(2);
    }
    return 0;
}
```

Save the program as dummy2.c. Compile and run it:

```
gcc -o dummy2 dummy2.c
./dummy2 &
```

Now, we can attach to dummy2 by using the code below:

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <linux/user.h>   /* For user_regs_struct
                             etc. */
int main(int argc, char *argv[])
{   pid_t traced_process;
    struct user_regs_struct regs;
    long ins;
    if(argc != 2) {
        printf("Usage: %s <pid to be traced>\n",
               argv[0], argv[1]);
        exit(1);
    }
    traced_process = atoi(argv[1]);
    ptrace(PTRACE_ATTACH, traced_process,
           NULL, NULL);
    wait(NULL);
    ptrace(PTRACE_GETREGS, traced_process,
           NULL, &regs);
    ins = ptrace(PTRACE_PEEKTEXT, traced_process,
                 regs.eip, NULL);
    printf("EIP: %lx Instruction executed: %lx\n",
           regs.eip, ins);
    ptrace(PTRACE_DETACH, traced_process,
           NULL, NULL);
    return 0;
}
```

The above program simply attaches to a process, waits for it to stop, examines its eip (instruction pointer) and detaches.

To inject code use ptrace(PTRACE_POKETEXT, ..) and ptrace(PTRACE_POKEDATA, ..) after the traced process has stopped.

Setting Breakpoints

How do debuggers set breakpoints? Generally, they replace the instruction to be executed with a trap instruction, so that when the traced program stops, the tracing program, the debugger, can examine it. It will replace the original instruction once the tracing program continues the traced process. Here's an example:

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <linux/user.h>
const int long_size = sizeof(long);
void getdata(pid_t child, long addr,
             char *str, int len)
{   char *laddr;
    int i, j;
    union u {
            long val;
            char chars[long_size];
    }data;
    i = 0;
    j = len / long_size;
    laddr = str;
    while(i < j) {
        data.val = ptrace(PTRACE_PEEKDATA, child,
                          addr + i * 4, NULL);
        memcpy(laddr, data.chars, long_size);
        ++i;
        laddr += long_size;
    }
    j = len % long_size;
    if(j != 0) {
        data.val = ptrace(PTRACE_PEEKDATA, child,
                          addr + i * 4, NULL);
        memcpy(laddr, data.chars, j);
    }
    str[len] = '\0';
}
void putdata(pid_t child, long addr,
             char *str, int len)
{   char *laddr;
    int i, j;
    union u {
            long val;
            char chars[long_size];
    }data;
    i = 0;
    j = len / long_size;
    laddr = str;
    while(i < j) {
        memcpy(data.chars, laddr, long_size);
        ptrace(PTRACE_POKEDATA, child,
               addr + i * 4, data.val);
        ++i;
        laddr += long_size;
    }
    j = len % long_size;
    if(j != 0) {
        memcpy(data.chars, laddr, j);
        ptrace(PTRACE_POKEDATA, child,
               addr + i * 4, data.val);
    }
}
int main(int argc, char *argv[])
{   pid_t traced_process;
    struct user_regs_struct regs, newregs;
    long ins;
    /* int 0x80, int3 */
    char code[] = {0xcd,0x80,0xcc,0};
    char backup[4];
    if(argc != 2) {
        printf("Usage: %s <pid to be traced>\n",
               argv[0], argv[1]);
        exit(1);
    }
    traced_process = atoi(argv[1]);
    ptrace(PTRACE_ATTACH, traced_process,
           NULL, NULL);
    wait(NULL);
    ptrace(PTRACE_GETREGS, traced_process,
           NULL, &regs);
    /* Copy instructions into a backup variable */
    getdata(traced_process, regs.eip, backup, 3);
    /* Put the breakpoint */
    putdata(traced_process, regs.eip, code, 3);
    /* Let the process continue and execute
       the int 3 instruction */
    ptrace(PTRACE_CONT, traced_process, NULL, NULL);
    wait(NULL);
    printf("The process stopped, putting back "
           "the original instructions\n");
    printf("Press <enter> to continue\n");
    getchar();
    putdata(traced_process, regs.eip, backup, 3);
```

```
        /* Setting the eip back to the original
           instruction to let the process continue */
        ptrace(PTRACE_SETREGS, traced_process,
               NULL, &regs);
        ptrace(PTRACE_DETACH, traced_process,
               NULL, NULL);
        return 0;
    }
```

Here we replace the three bytes with the code for a trap instruction, and when the process stops, we replace the original instructions and reset the eip to original location. Figures 1-4 clarify how the instruction stream looks when above program is executed.

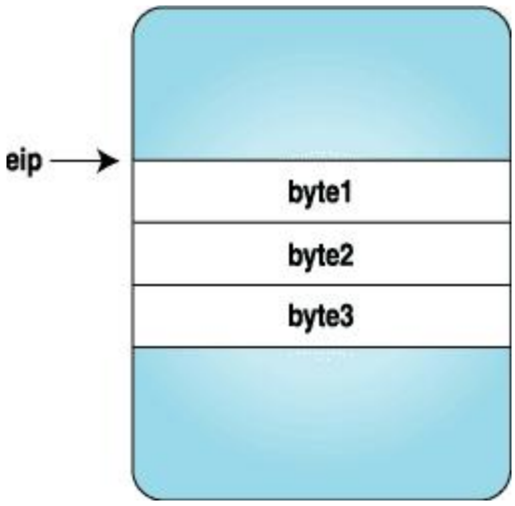
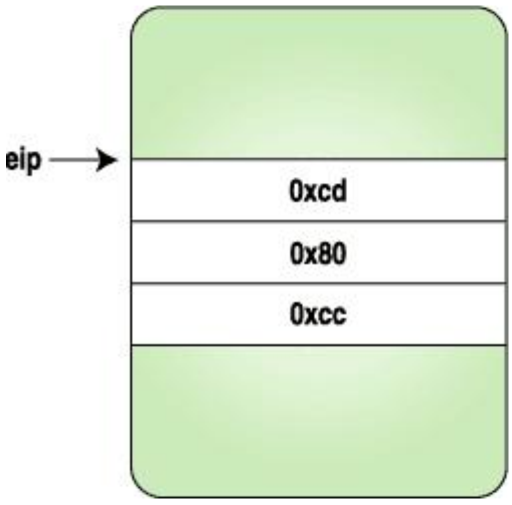
	

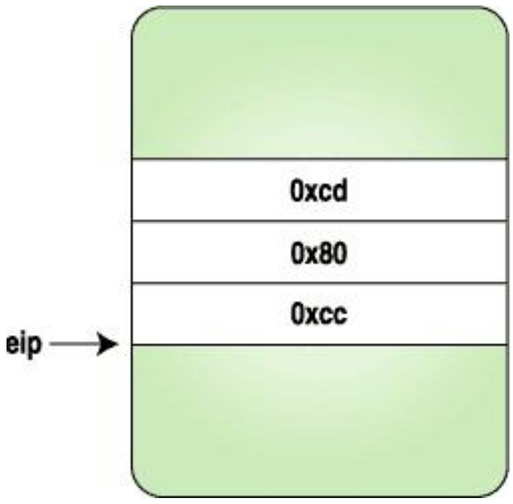
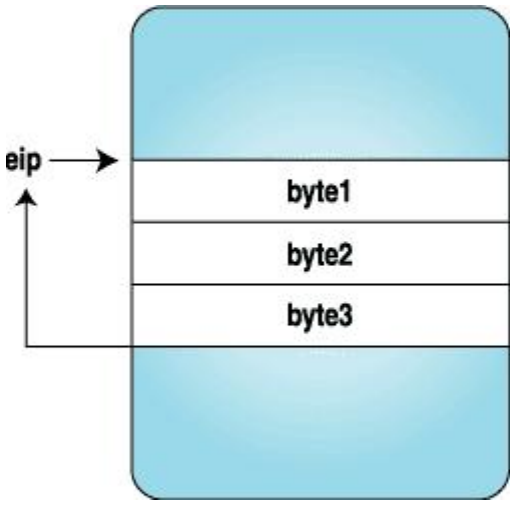
Figure 1. After the Process Is Stopped	Figure 2. After the Trap Instruction Bytes Are Set
	

Figure 3. Trap Is Hit and Control Is Given to the Tracing Program	Figure 4. After the Original Instructions Are Replaced and eip Is Reset to the Original Location

Now that we have a clear idea of how breakpoints are set, let's inject some code bytes into a running program. These code bytes will print “hello world”.

The following program is a simple “hello world” program with modifications to fit our needs. Compile the following program with:

```
gcc -o hello hello.c
void main()
{
__asm__("
         jmp forward
backward:
         popl   %esi      # Get the address of
                          # hello world string
         movl   $4, %eax  # Do write system call
         movl   $2, %ebx
         movl   %esi, %ecx
         movl   $12, %edx
         int    $0x80
         int3             # Breakpoint. Here the
                          # program will stop and
                          # give control back to
                          # the parent
forward:
         call   backward
         .string \"Hello World\\n\""
       );
}
```

The jumping backward and forward here is required to find the address of the “hello world” string.

We can get the machine code for the above assembly from GDB. Fire up GDB and disassemble the program:

```
(gdb) disassemble main
Dump of assembler code for function main:
0x80483e0 <main>:       push   %ebp
0x80483e1 <main+1>:     mov    %esp,%ebp
0x80483e3 <main+3>:     jmp    0x80483fa <forward>
End of assembler dump.
(gdb) disassemble forward
Dump of assembler code for function forward:
0x80483fa <forward>:    call   0x80483e5 <backward>
0x80483ff <forward+5>:  dec    %eax
0x8048400 <forward+6>:  gs
0x8048401 <forward+7>:  insb   (%dx),%es:(%edi)
0x8048402 <forward+8>:  insb   (%dx),%es:(%edi)
0x8048403 <forward+9>:  outsl  %ds:(%esi),(%dx)
0x8048404 <forward+10>: and    %dl,0x6f(%edi)
0x8048407 <forward+13>: jb     0x8048475
0x8048409 <forward+15>: or     %fs:(%eax),%al
0x804840c <forward+18>: mov    %ebp,%esp
0x804840e <forward+20>: pop    %ebp
0x804840f <forward+21>: ret
End of assembler dump.
(gdb) disassemble backward
Dump of assembler code for function backward:
0x80483e5 <backward>:   pop    %esi
```

```
0x80483e6 <backward+1>: mov    $0x4,%eax
0x80483eb <backward+6>: mov    $0x2,%ebx
0x80483f0 <backward+11>:        mov    %esi,%ecx
0x80483f2 <backward+13>:        mov    $0xc,%edx
0x80483f7 <backward+18>:        int    $0x80
0x80483f9 <backward+20>:        int3
End of assembler dump.
```

We need to take the machine code bytes from main+3 to backward+20, which is a total of 41 bytes. The machine code can be seen with the x command in GDB:

```
(gdb) x/40bx main+3
<main+3>: eb 15 5e b8 04 00 00 00
<backward+6>: bb 02 00 00 00 89 f1 ba
<backward+14>: 0c 00 00 00 cd 80 cc
<forward+1>: e6 ff ff ff 48 65 6c 6c
<forward+9>: 6f 20 57 6f 72 6c 64 0a
```

Now we have the instruction bytes to be executed. Why wait? We can inject them using the same method as in the previous example. The following is the source code; only the main function is given here:

```
int main(int argc, char *argv[])
{   pid_t traced_process;
    struct user_regs_struct regs, newregs;
    long ins;
    int len = 41;
    char insertcode[] =
"\xeb\x15\x5e\xb8\x04\x00"
        "\x00\x00\xbb\x02\x00\x00\x00\x89\xf1\xba"
        "\x0c\x00\x00\x00\xcd\x80\xcc\xe8\xe6\xff"
        "\xff\xff\x48\x65\x6c\x6c\x6f\x20\x57\x6f"
        "\x72\x6c\x64\x0a\x00";
    char backup[len];
    if(argc != 2) {
        printf("Usage: %s <pid to be traced>\n",
               argv[0], argv[1]);
        exit(1);
    }
    traced_process = atoi(argv[1]);
    ptrace(PTRACE_ATTACH, traced_process,
           NULL, NULL);
    wait(NULL);
    ptrace(PTRACE_GETREGS, traced_process,
           NULL, &regs);
    getdata(traced_process, regs.eip, backup, len);
    putdata(traced_process, regs.eip,
            insertcode, len);
    ptrace(PTRACE_SETREGS, traced_process,
           NULL, &regs);
    ptrace(PTRACE_CONT, traced_process,
           NULL, NULL);
    wait(NULL);
    printf("The process stopped, Putting back "
           "the original instructions\n");
    putdata(traced_process, regs.eip, backup, len);
    ptrace(PTRACE_SETREGS, traced_process,
           NULL, &regs);
    printf("Letting it continue with "
           "original flow\n");
    ptrace(PTRACE_DETACH, traced_process,
           NULL, NULL);
    return 0;
}
```

Injecting the Code into Free Space

In the previous example we injected the code directly into the executing instruction stream. However, debuggers can get confused with this kind of behaviour, so let's find the free space in the process and inject the code there. We can find free space by examining the /proc/pid/maps file of the traced process. The following function will find the starting address of this map:

```
long freespaceaddr(pid_t pid)
{
    FILE *fp;
    char filename[30];
    char line[85];
    long addr;
    char str[20];
    sprintf(filename, "/proc/%d/maps", pid);
    fp = fopen(filename, "r");
    if(fp == NULL)
        exit(1);
    while(fgets(line, 85, fp) != NULL) {
        sscanf(line, "%lx-%*lx %*s %*s %s", &addr,
               str, str, str, str);
        if(strcmp(str, "00:00") == 0)
            break;
    }
    fclose(fp);
    return addr;
}
```

Each line in /proc/pid/maps represents a mapped region of the process. An entry in /proc/pid/maps looks like this:

```
map start-mapend    protection  offset     device
inode      process file
08048000-0804d000    r-xp        00000000   03:08
66111      /opt/kde2/bin/kdeinit
```

The following program injects code into free space. It's similar to the previous injection program except the free space address is used for keeping our new code. Here is the source code for the main function:

```
int main(int argc, char *argv[])
{   pid_t traced_process;
    struct user_regs_struct oldregs, regs;
    long ins;
    int len = 41;
    char insertcode[] =
"\xeb\x15\x5e\xb8\x04\x00"
        "\x00\x00\xbb\x02\x00\x00\x00\x89\xf1\xba"
        "\x0c\x00\x00\x00\xcd\x80\xcc\xe8\xe6\xff"
        "\xff\xff\x48\x65\x6c\x6c\x6f\x20\x57\x6f"
        "\x72\x6c\x64\x0a\x00";
    char backup[len];
    long addr;
    if(argc != 2) {
        printf("Usage: %s <pid to be traced>\n",
               argv[0], argv[1]);
        exit(1);
    }
    traced_process = atoi(argv[1]);
    ptrace(PTRACE_ATTACH, traced_process,
           NULL, NULL);
    wait(NULL);
    ptrace(PTRACE_GETREGS, traced_process,
           NULL, &regs);
    addr = freespaceaddr(traced_process);
    getdata(traced_process, addr, backup, len);
```

```
       putdata(traced_process, addr, insertcode, len);
       memcpy(&oldregs, &regs, sizeof(regs));
       regs.eip = addr;
       ptrace(PTRACE_SETREGS, traced_process,
              NULL, &regs);
       ptrace(PTRACE_CONT, traced_process,
              NULL, NULL);
       wait(NULL);
       printf("The process stopped, Putting back "
              "the original instructions\n");
       putdata(traced_process, addr, backup, len);
       ptrace(PTRACE_SETREGS, traced_process,
              NULL, &oldregs);
       printf("Letting it continue with "
              "original flow\n");
       ptrace(PTRACE_DETACH, traced_process,
              NULL, NULL);
       return 0;
   }
```

Behind the Scenes

So what happens within the kernel now? How is ptrace implemented? This section could be an article on its own; however, here's a brief description of what happens.

When a process calls ptrace with PTRACE_TRACEME, the kernel sets up the process flags to reflect that it is being traced:

```
   Source: arch/i386/kernel/ptrace.c
   if (request == PTRACE_TRACEME) {
       /* are we already being traced? */
       if (current->ptrace & PT_PTRACED)
           goto out;
       /* set the ptrace bit in the process flags. */
       current->ptrace |= PT_PTRACED;
       ret = 0;
       goto out;
   }
```

When a system call entry is done, the kernel checks this flag and calls the trace system call if the process is being traced. The gory assembly details can be found in arch/i386/kernel/entry.S.

Now, we are in the sys_trace() function as defined in arch/i386/kernel/ptrace.c. It stops the child and sends a signal to the parent notifying that the child is stopped. This wakes up the waiting parent, and it does the ptrace magic. Once the parent is done, and it calls ptrace(PTRACE_CONT, ..) or ptrace(PTRACE_SYSCALL, ..), it wakes up the child by calling the scheduler function wake_up_process(). Some other architectures can implement this by sending a SIGCHLD to child.

Conclusion

ptrace may appear to be magic to some people, because it can examine and modify a running program. It is generally used by debuggers and system call tracing programs, such as ptrace. It opens up interesting possibilities for doing user-mode extensions as well. There have been a lot of attempts to extend the operating system on the user level. See Resources to read about UFO, a user-level extension to filesystems. **ptrace** also is used to employ security mechanisms.

All example code from this article and from Part I is available as a tar archive on the *Linux Journal* FTP site [ftp.linuxjournal.com/pub/lj/listings/issue104/6210.tgz].

Resources

email: ppadala@cise.ufl.edu

Pradeep Padala is currently working on his Master's degree at the University of Florida. His research interests include Grid and distributed systems. He can be reached via e-mail at p_padala@yahoo.com or through his web site (www.cise.ufl.edu/~ppadala).

Linux Powers Four-Wall 3-D Display

Douglas B. Maxwell

Issue #104, December 2002

Replacing a visualization supercomputing platform driving a four-wall immersive display system with a PC commodity cluster.

The cost per performance of PC clusters is making them a viable alternative to traditional high-end visualization supercomputers. Rapid evolution in commodity PC hardware has both driven down costs and accelerated obsolescent computing cycles. A general rule to follow for buying graphics capability from SGI is to budget $250,000 US per graphics pipe. For those who cannot afford to outfit an Onyx-class computer with four graphics pipes, extra raster managers may be added to two pipes to drive a four-wall display system. In contrast, our experimental cluster costs less than $1,000 US per node. With the addition of a video matrix switcher, the grand total was less than $15,000 US. Even with the fast obsolescence cycles, the price difference is so great that an organization could afford to replace or upgrade the graphics clusters many times during the lifetime of one SGI system. Another advantage of PCs is the wide availability of low-cost parts. Overall, the PCs are cost effective, powerful and flexible.

We present an experiment that integrates a commodity cluster into an existing four-wall display system—a Surround-Screen Visualization System (SSVR) from Mechdyne Corporation. The objective is to attain active stereo visualization on multiple walls using genlocking, swap-locking and data-locking capabilities.

High-end visualization supercomputers offer multiwall, active stereo visualization packaged together. Stereo presentation and coordination of scene graph data is taken care of automatically by the computer in hardware or by the invocation of proprietary software libraries. The cluster was designed from the beginning to attempt to replace aging SGI computing equipment used to drive our current four-wall display system. We are finding ourselves taxing the capabilities of an Onyx 2 system with Infinite Reality 2 graphics by demanding increasing numbers of polygons to be rendered while needing a fixed frame rate for active stereo. When implementing a cluster, these issues must be dealt with in order to produce a coherent scene across many screens.

Communication between the cluster nodes is vital. Data such as pixels, geometric primitives or even scene graph data is passed among the nodes. The way data is handled and the type of data passed greatly impacts the network bandwidth requirements of the cluster. Two basic approaches for setting up a graphics clustering communication software architecture are client/server and master/slave.

In the client/server approach, a single node serves data to the graphics rendering clients. The advantage to this arrangement is many applications may embed a server that works with the same rendering client nodes. This environment is very flexible. The disadvantage is a higher consumption of network bandwidth. Most client/server clusters rely upon relatively expensive Myrinet or gigabit Ethernet hardware.

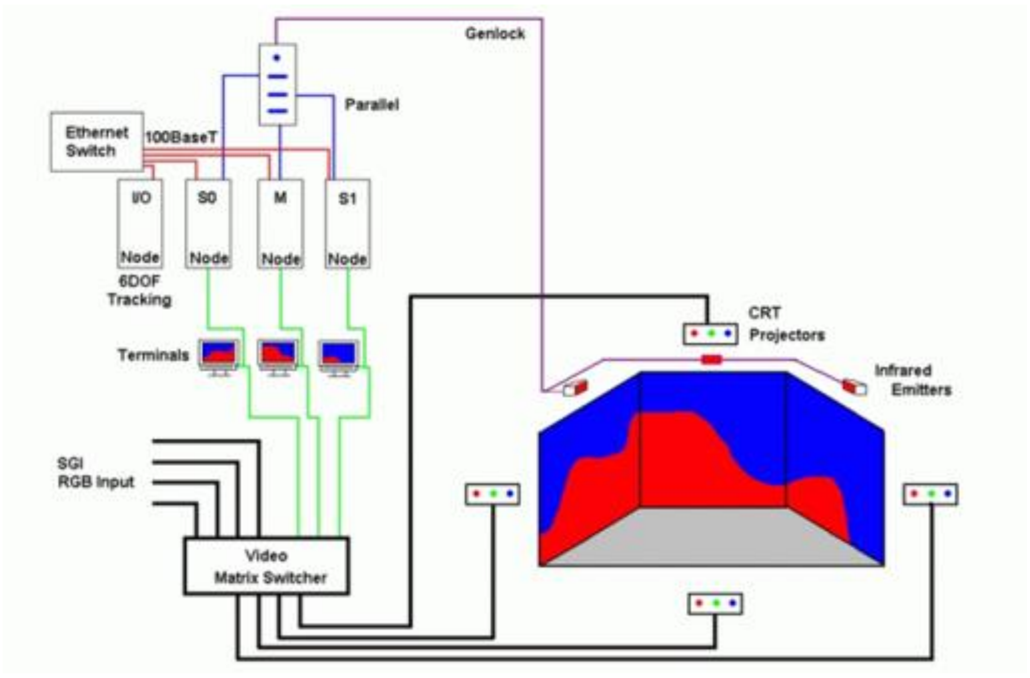


Figure 1. Commodity Cluster Configuration Diagram

The master/slave approach, used in this project, consists of multiple nodes, where each node of the graphics cluster locally stores and runs an identical copy of the graphics application. Consequently, only a small amount of information is required to be shared among the nodes, and network bandwidth becomes less of a concern. This information may simply include input device data and timestamps. In this configuration, the master node handles application state changes.

All graphics clusters must satisfy the following three requirements:

- Genlocking: the process of synchronizing the video frames from each node in a cluster so that they produce a fluid, coherent image. Genlocking may be achieved through software or hardware.
- Swap Locking: the process of synchronizing the frame buffer rendering and swapping. This is necessary because each view of a scene contains different amounts of data and numbers of polygons to render. These may produce different rendering times for each frame for each node.
- Data Locking: the process of synchronizing the views to maintain consistency across the screens. This becomes an issue since each node in the cluster renders its frames from locally stored information.

We used a set of standard PC configurations equipped with MSI G4Ti4600 graphics adapters powered by NVIDIA's GeForce4 Ti graphics processing unit (GPU) and 128Mb of DDR video memory. Although not completely necessary, the PCs were identical, which made software installation easier. The PCs communicated via 100BaseT networking adapters and a 100BaseT switch.

The projectors of the SSVR are connected to an Extron CrossPoint Plus 124 matrix video switcher. The switcher is capable of accepting video input from 12 sources and output to four sources.

Since genlocking and data locking are handled in software through the parallel ports, a special box (Figure 2) was fabricated to handle the signaling appropriately. This box was also built from commercial off-the-shelf hardware for less than $20 US. Besides controlling the switcher, this box also outputs a genlocking signal to a set of Crystal Eyes infrared emitters.

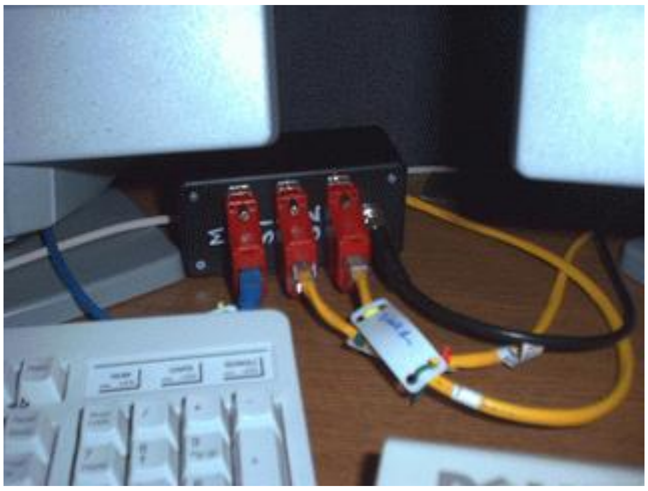


Figure 2. A fabricated box that handles genlocking and data locking through parallel port connections.

System Implementation

A variety of software is used to deal with the active stereo and scene synchronization needs of the cluster. The commodity cluster was built upon a standard installation of Red Hat Linux 7.2, and the kernel was then patched to include the Real-Time Application Interface (RTAI), which allows for low latency and task completion timing.

SoftGenLock and RTAI are used in concert to provide a software active stereo solution. The RTAI kernel module detects the vertical refresh of the monitor and changes a pointer in the video card memory that tells the video card what to draw on the screen. A double-sized buffer is provided to the application by specifying a virtual frame buffer in XFree86, which is twice the size of the actual frame buffer being drawn to the screen. For example, if displaying at a resolution of 1024 × 768 in active stereo, the virtual desktop would need to be at a resolution of 2048 × 768. The RTAI kernel module splits the frame buffer in half and alternatingly displays the scene in 1024 × 768 pieces. An application draws in stereo by rendering to the right and left sides of the X frame buffer for the right and left eye.

Genlock/data lock is achieved by synchronizing the machines in the cluster over the parallel ports. The RTAI kernel module writes to one pin on the parallel port and reads from another to make sure that the other machines in the cluster have completed a frame. The master tells all the other nodes in the cluster when to draw, and the other nodes in the cluster report back when they are ready for a new frame. Data lock is achieved by making sure that the slave machines in the cluster are on the correct eye when the parallel port is in a certain state.

SoftGenLock does not synchronize applications between the nodes of a cluster; it provides data lock and stereo. It is the responsibility of the application to synchronize the viewing frustum and animations in the application. Lastly, since SoftGenLock only uses VGA registers, it potentially can work with any graphics card.

When installing a PC cluster, a lot of issues must be dealt with before setup. Here are the important points and some recommendations on how to set up the system properly:

- Be sure to have adequate HVAC, power and network access where the cluster will be set up.
- Set up the cluster and display system in separate rooms, because noise levels from fans, drives, etc., may be distracting.

- Prepare to deal with a lot of issues associated with laying down the cables. The time taken at this stage to label and check the cabling for proper size and lengths will save time later. This will alleviate problems with signal degradation and simply losing a cable in the “nest”.
- Create a master power switch to turn off all units at once.

We constructed and installed a three-wall cluster in less than two weeks. This cluster was configured to use active stereo and has the ability to demonstrate swap locking and data locking.

Testing and Evaluation

We tested the PC commodity cluster by implementing a simple visualization software system. We utilized an interoperable software architecture, VR Juggler, which provides a set of programming abstractions for interfacing with a variety of display, tracking and computing systems and a variety of interaction devices. The software, which is used for visualizing terrain information, has a variety of display modes including stereo. It can be recompiled to work on different computing architectures and reconfigured at execution time by including different default device configuration files.

The application integrated smoothly with the cluster system and performed better than expected. The quality of the displays and stereo viewing was comparable to the same software running on three walls using an SGI Onyx 2 with IR2 graphics. However, the cluster was able to visualize the data with better performance. Figure 3 shows the cluster software running in the four-wall display system on the three side walls.

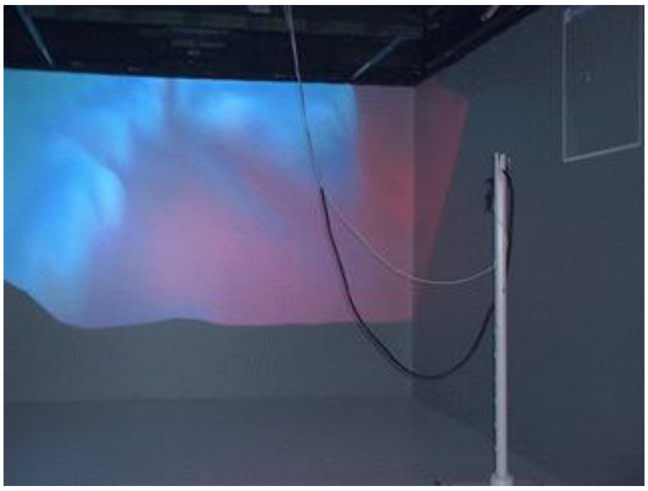


Figure 3. A Test Application Driving a Four-Wall Display

Summary and Conclusions

The use of a graphics PC cluster is now becoming a viable low-cost alternative to the use of single large visualization supercomputers. The PCs and related video hardware are fairly cheap, computationally powerful and flexible. There also is an abundance of software available for them, such as VR Juggler, which allow for applications to interface with a variety of display systems and interaction devices easily. Our cluster experiment explored the feasibility of phasing out and replacing existing expensive single large computing hardware. The results show we can make this kind of transition in the near future, and we believe our experiences will be motivation for others to follow suit.

Acknowledgements

The author would like to thank the following people for their support and contributions to the project: Aaron Bryden, Greg S. Schmidt, Ian Roth and J. Edward Swan II.

Resources

email: maxwelldb@npt.nuwc.navy.mil

Douglas B. Maxwell (vrdeity@yahoo.com) is a mechanical engineer and recently transferred to the Naval Undersea Warfare Center's Weapons and Countermeasure Control group in Newport, Rhode Island. His previous post was the Naval Research Laboratory Virtual Reality Lab in Washington, DC. He specializes in design synthesis, simulation and training in virtual environments.

Learning the iTunesDB File Format

Patrick Crosby

Issue #104, December 2002

Getting into the sequences, byte types and math of the iPod MP3 player.

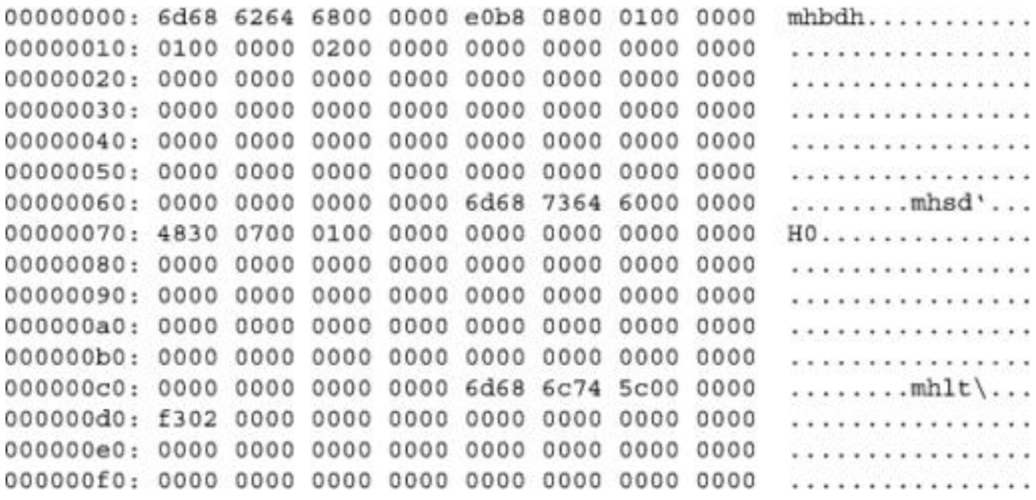
Reverse engineering is one of the skills necessary on many Linux software projects, because many file formats have no open specifications. Even if we have a specification, we will probably have to reverse engineer certain portions to learn all the details. Reverse engineering is a rewarding experience. Not only do we learn something in detail, but it is satisfying to offer Linux users access to previously unsupported devices.

This article discusses the specific example of reverse engineering the iTunesDB file format used by the Apple iPod portable MP3 player. Although this example covers a file format, we can use similar techniques for device drivers and network streams.

The key strategies for reverse engineering are hypothesis, pattern recognition and validation. We use hypotheses to guess what will be in the file. All files contain repeated patterns, and recognizing those patterns is the key to reverse engineering. Validation consists of developing software to check whether a file conforms to a format. If we can tell that a file is valid, we understand the format.

We need a hypothesis for the types of information an unknown file format contains. We know that an iPod plays digital music and the songs can be organized into playlists. Because we can't find any other files on the iPod hard disk for storing this information, we guess that it will be in the iTunesDB file. The goal of trying to find song and playlist information narrows our search when we look for patterns in the file.

Although every file format contains repeated patterns, the trick is recognizing these patterns and learning how to parse them. Opening a file in a hex editor allows us to examine the raw data for patterns. Figure 1 contains the first 256 bytes of an iTunesDB file as shown by Emacs in hexl-mode. The first column is the file position; then there are eight columns, each column containing two bytes. The final column is the ever-helpful ASCII representation of the data.



```
00000000: 6d68 6264 6800 0000 e0b8 0800 0100 0000  mhbdh...........
00000010: 0100 0000 0200 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 6d68 7364 6000 0000  ........mhsd`...
00000070: 4830 0700 0100 0000 0000 0000 0000 0000  H0..............
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 6d68 6c74 5c00 0000  ........mhlt\...
000000d0: f302 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Figure 1. First 256 Bytes of the iTuneDB File

From only the first 256 bytes, a pattern emerges: three 5-byte sequences start with “mh” (0x6d68). When we scan the rest of the file, we find it littered with 5-byte mh sequences (see Figure 2). Based on the number of mh byte sequences we see in the file, we hypothesize that these sequences are a marker for a data structure. We don't know if the data structure has a fixed or varying size. Perhaps it is only five bytes long and the information between the mh sequences is composed of other data structures. To help us determine this information, we will extract the first 15 mh sequences from the file (Figure 3).

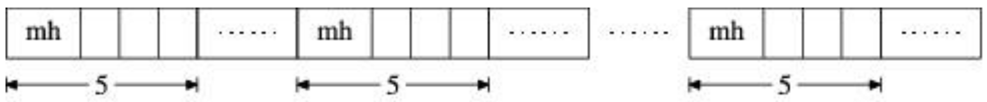
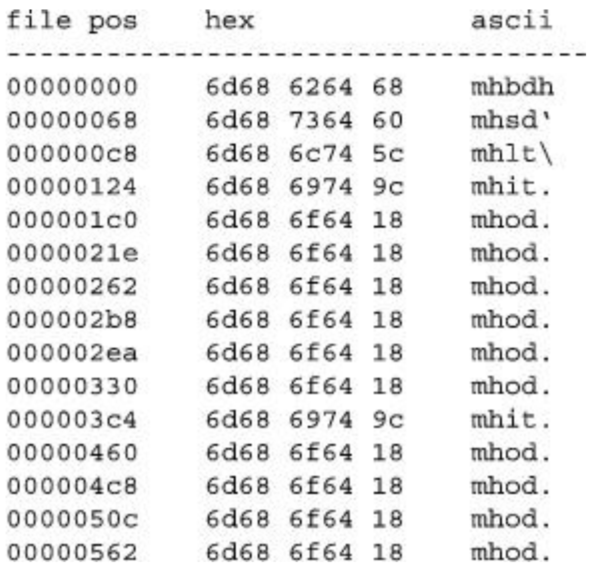


Figure 2. 5-Byte mh Sequences



```
file pos    hex            ascii
-----------------------------------
00000000    6d68 6264 68   mhbdh
00000068    6d68 7364 60   mhsd`
000000c8    6d68 6c74 5c   mhlt\
00000124    6d68 6974 9c   mhit.
000001c0    6d68 6f64 18   mhod.
0000021e    6d68 6f64 18   mhod.
00000262    6d68 6f64 18   mhod.
000002b8    6d68 6f64 18   mhod.
000002ea    6d68 6f64 18   mhod.
00000330    6d68 6f64 18   mhod.
000003c4    6d68 6974 9c   mhit.
00000460    6d68 6f64 18   mhod.
000004c8    6d68 6f64 18   mhod.
0000050c    6d68 6f64 18   mhod.
00000562    6d68 6f64 18   mhod.
```

Figure 3. First 15 mh Sequences

The numbers 0x9c and 0x18 do not translate to "." using ASCII (the ASCII code for . is 0x2e). So, we follow the Emacs example and use . for anything out of the range of standard ASCII characters.

Although we have only a few data points, we find multiple potential patterns. Certain sequences are repeated, such as "mhod." and "mhit.". In fact, both of the mhit. sequences are followed by mhod. sequences. Perhaps the patterns exist throughout the file. Before sifting through hundreds of kilobytes in a hex editor, it is a good idea to consider using the third reverse engineering strategy: validation.

A great way to test out hypotheses is to write a validation program, and the sooner the better. The goal of the validation program is to parse the file and determine if it is valid using various tests derived from pattern recognitions and hypotheses. The first iterations of the validation program will have few tests (if any) that are correct, but as we come to understand the format, the program evolves into a compliance tester. It is then useful for checking software that modifies files of this format, because we can use it to ensure that our changes are valid.

Our first validation program will display information (shown in Figure 4) about all the mh chunks in the file to help us find patterns. The first glaring pattern we see pertains to the difference between the file positions of the mh chunks. With the exception of the mhod chunks, the fifth byte contains the number of bytes until the next mh chunk. We can add code to our validation program to check this pattern, and we find that this is almost the case: there are a few chunks besides the mhod chunks that fail our test. It is unlikely that the creators of the iTunesDB file format would limit the size of a data structure to 256 bytes, thus we are probably dealing with a multibyte number.

```
file pos    hex           ascii   prev pos diff
-----------------------------------------------------
00000000    6d68 6264 68  mhbdh   0x0
00000068    6d68 7364 60  mhsd'   0x68
000000c8    6d68 6c74 5c  mhlt\   0x60
00000124    6d68 6974 9c  mhit.   0x5c
000001c0    6d68 6f64 18  mhod.   0x9c
0000021e    6d68 6f64 18  mhod.   0x5e
00000262    6d68 6f64 18  mhod.   0x44
000002b8    6d68 6f64 18  mhod.   0x56
000002ea    6d68 6f64 18  mhod.   0x32
00000330    6d68 6f64 18  mhod.   0x46
000003c4    6d68 6974 9c  mhit.   0x94
00000460    6d68 6f64 18  mhod.   0x9c
000004c8    6d68 6f64 18  mhod.   0x68
0000050c    6d68 6f64 18  mhod.   0x44
00000562    6d68 6f64 18  mhod.   0x56
(etc.)

unique sequences:
hex           ascii   # of occurrences
----------------------------------------------
6d68 6264 68  mhbdh   1
6d68 7364 60  mhsd'   2
6d68 6c74 5c  mhlt\   1
6d68 6974 9c  mhit.   755
6d68 6f64 18  mhod.   5297
6d68 6c70 5c  mhlp\   1
6d68 7970 6c  mhypl   4
6d68 6970 4c  mhipL   807
```

Figure 4. Results of Validation Program

As soon as the word “multibyte” crosses our mind, we must consider the endianness of the file. If this first byte is part of a multibyte number, it is the least significant byte, thus a little endian number. If one number in the file is little-endian, the whole file will be little-endian. Only a sadistic designer would create a mixed-endian file format. Bytes six, seven and eight are almost always zero, so we guess the chunk size is a 4-byte number.

Figure 5 contains a diagram of our theory for the basic building block of the file format. When we change our validation program to use a 4-byte number for the chunk size, the test is successful for all non-mhod chunks.

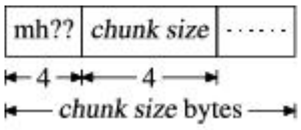


Figure 5. Theory of File Format

Reverse engineering requires us to oscillate between our three strategies in order to decipher the file format. We will continue to employ these strategies to discover more details. We wrote software that can parse the file format into data chunks of variable size and type. We would like to know what every byte in the chunk signifies, and this requires more hypotheses, pattern recognitions and validations.

We start at the top of the file and focus on the mhbd chunk. Given its frequency and its location at the start of the file, it probably contains header information for the whole file. We know it is 0x68 bytes long, and we accounted for only eight of those bytes with our standard tag and chunk length fields. Everything else thus far consists of 4-byte numbers and we have a multiple of four bytes left, so we will break up the remaining 96 bytes into 24 4-byte numbers, as shown in Figure 6.

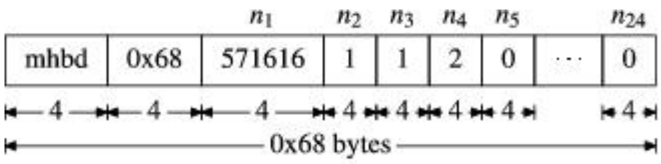
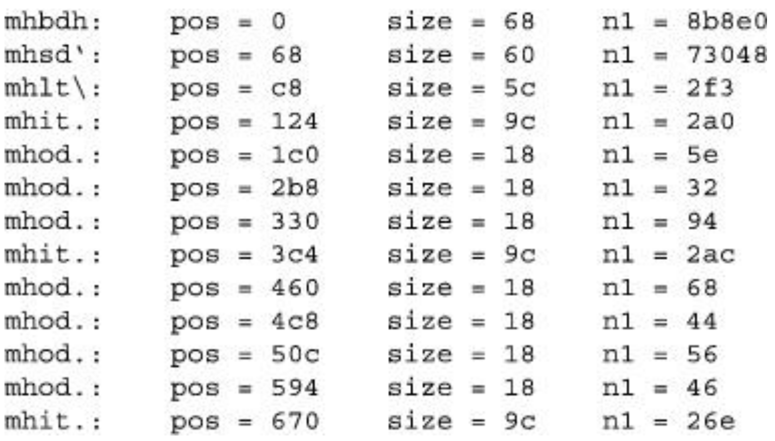


Figure 6. The Remaining 96 Bytes

The size in bytes of the example iTunesDB file is 571,616. We can add a test to the validation program and verify that n1 is the file size for all iTunesDB files. The n2, n3, n4 trio resembles a version number (1.1.2). Looking at multiple iTunesDB files teaches us that n2 through n4 changes to 1.2.2 for iTunesDB files from newer iPods. n5 to n24 are always zero.

Storing the file size in the header is a curious action. We determined a consistent pattern for the first eight bytes of an mh chunk; perhaps bytes 9-12 are consistent for all mh chunks. We modify our validation program to spit out the value of bytes 9-12 for all mh chunks (see Figure 7 for results).



```
mhbdh:    pos = 0       size = 68     n1 = 8b8e0
mhsd':    pos = 68      size = 60     n1 = 73048
mhlt\:    pos = c8      size = 5c     n1 = 2f3
mhit.:    pos = 124     size = 9c     n1 = 2a0
mhod.:    pos = 1c0     size = 18     n1 = 5e
mhod.:    pos = 2b8     size = 18     n1 = 32
mhod.:    pos = 330     size = 18     n1 = 94
mhit.:    pos = 3c4     size = 9c     n1 = 2ac
mhod.:    pos = 460     size = 18     n1 = 68
mhod.:    pos = 4c8     size = 18     n1 = 44
mhod.:    pos = 50c     size = 18     n1 = 56
mhod.:    pos = 594     size = 18     n1 = 46
mhit.:    pos = 670     size = 9c     n1 = 26e
```

Figure 7. Value of Bytes 9-12

When we add n1 to the file position of the mhbd chunk, we get to the end of the file. Adding n1 to the mhod chunk position is the position of the next mh chunk. For mhsd, adding 0x73048 to 0x68 is 0x730b0. Going to that file position in our hex editor shows us that at 0x730b0 is the start of another mhsd chunk.

The n1 plus file position value for mhit, mhyp and mhip chunks is consistent with the one for mhbd, mhsd and mhod. Figure 8 contains an example of the relationship between these chunks. For mhlt and mhlp, the position at n1 plus the mhlt/mhlp file position is in the middle of a chunk. The n1 value for mhlt in this example is 755, which is the number of mhit chunks in the file. The n1 value for mhlp is 4, which is the number of mhyp chunks. These relationships suggest a nested data structure. If we use the n1 size to determine parent-child relationships between mh chunks, we can create the tree diagram shown in Figure 9. The symmetry looks too good to be wrong, so we'll stick with it as our theory unless we find data later on that convinces us otherwise.

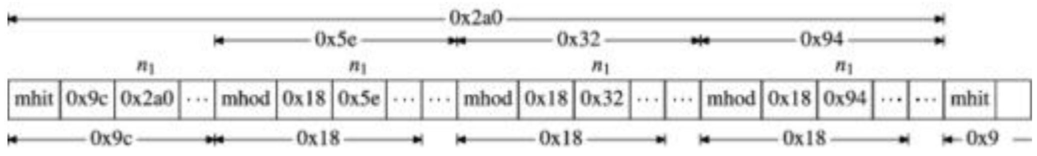


Figure 8. The Relationship between the mhit, mhyp and mhip Chunks and the mhbd, mhsd and mhod Chunks

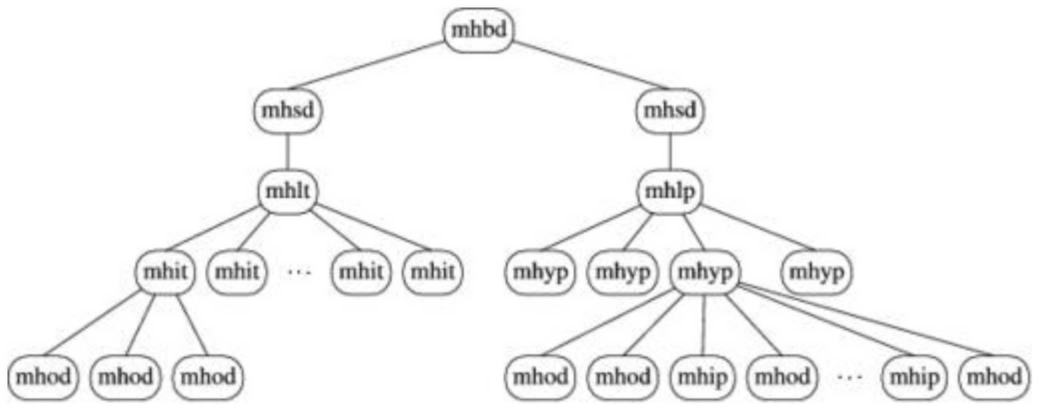


Figure 9. Parent-Child mh Chunks

Going down the tree one level from mhbd, we have a pair of mhsd nodes, each 104 bytes long. Breaking the data after the initial 12 bytes into 4-byte numbers shows us that only the first 4-byte number contains any non-null data. It equals one when the child node is mhlt and two when the child node is mhlp. Figure 10 contains a description of the mhsd chunk.

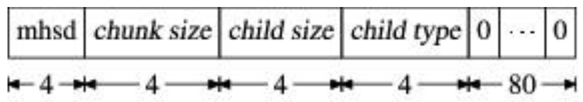


Figure 10. The mhsd Chunk

Continuing down to the next level, we find the mhlt and mhlp nodes. We can verify that following the number of children they end with 80 null bytes, as shown in Figure 11.

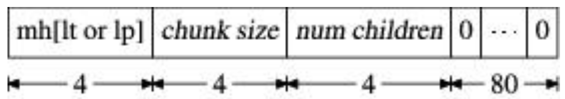


Figure 11. The mhlt and mhlp Nodes

The children of the mhlt node are mhit nodes. The number of mhit nodes corresponds to the number of songs on the iPod. Dividing the first mhit chunk into 4-byte unsigned integers yields the diagram in Figure 12.

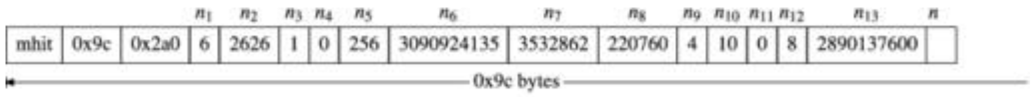


Figure 12. Dividing the First mhit Chunk into 4-byte Unsigned Integers

The only number that looks at all familiar simply by its magnitude is n6. In UNIX, time is usually determined as the number of seconds since the Epoch (00:00:00 UTC, January 1, 1970). In Apple's HFS+ filesystem, however, time is stored as the number of seconds since 00:00:00 UTC, January 1, 1904. Converting 3090924135 to a date yields December 11, 2001, 06:02:15. We can't make much sense of the rest of these numbers yet, but by looking at many mhit chunks we notice that n1 equals the number of mhod children, n3 is always one, and n4 is always zero.

Perhaps by looking at the mhod children of the mhit chunks we can learn what the unknown mhit values signify. Indeed, a cursory look at some mhod chunks (see Figure 13) indicates they contain variable-length text after them.

```
000001c0: 6d68 6f64 1800 0000 5e00 0000 0100 0000  mhod....^.......
000001d0: 0000 0000 0000 0000 0100 0000 3600 0000  ............6...
000001e0: 0000 0000 0000 0000 4400 6500 6100 6400  ........D.e.a.d.
000001f0: 2000 4100 6e00 6700 6500 6c00 7300 2000   .A.n.g.e.l.s. .
00000200: 4d00 6100 6b00 6500 2000 5300 6c00 6f00  M.a.k.e. .S.l.o.
00000210: 7700 2000 5300 6f00 7500 6e00 6400 6d68  w. .S.o.u.n.d.mh
00000220: 6f64 1800 0000 4400 0000 0400 0000 0000  od....D.........
00000230: 0000 0000 0000 0100 0000 1c00 0000 0000  ................
00000240: 0000 0000 0000 4400 6500 7400 6100 6300  ......D.e.t.a.c.
00000250: 6800 6d00 6500 6e00 7400 2000 4b00 6900  h.m.e.n.t. .K.i.
00000260: 7400 6d68 6f64 1800 0000 5600 0000 0300  t.mhod....V.....
00000270: 0000 0000 0000 0000 0000 0100 0000 2e00  ................
00000280: 0000 0000 0000 0000 0000 5400 6800 6500  ..........T.h.e.
00000290: 7900 2000 5200 6100 6700 6900 6e00 6700  y. .R.a.g.i.n.g.
000002a0: 2c00 2000 5100 7500 6900 6500 7400 2000  ,. .Q.u.i.e.t. .
000002b0: 4100 7200 6d00 7900 6d68 6f64 1800 0000  A.r.m.y.mhod....
000002c0: 3200 0000 0500 0000 0000 0000 0000 0000  2...............
000002d0: 0100 0000 0a00 0000 0000 0000 0000 0000  ................
000002e0: 4900 6e00 6400 6900 6500 6d68 6f64 1800  I.n.d.i.e.mhod..
000002f0: 0000 4600 0000 0600 0000 0000 0000 0000  ..F.............
00000300: 0000 0100 0000 1e00 0000 0000 0000 0000  ................
00000310: 0000 4d00 5000 4500 4700 2000 6100 7500  ..M.P.E.G. .a.u.
00000320: 6400 6900 6f00 2000 6600 6900 6c00 6500  d.i.o. .f.i.l.e.
```

Figure 13. mhod Children of mhit Chunks

We know that n1 of the mhod chunk (see Figure 14) tells us how long the mhod chunk is including the text data. We need a hypothesis for what the remaining 12 bytes contain.

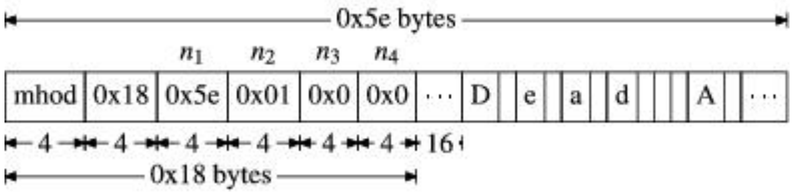


Figure 14. n1 of mhod Chunks

By reading the text, we see the artist, album and title strings are included, among others. The software needs to know what type of string it is for the string to be useful. This information could be provided by always having the strings in the same order: album, title, artist, genre and so on, but what if a song didn't have an album? It is more likely, then, that the type of string is encoded in the chunk somewhere. n2 looks like a good candidate. Looking at the strings and the n2 value, we determine the relationship shown in Table 1.

<u>Table 1. The n2 Value and Description</u>

The text data is not pure ASCII because the letters (in this example) are separated by a null byte. As there are two bytes for every letter, we assume we are dealing with a multibyte representation of text, most likely a Unicode encoding. There are 16 bytes before the text starts, however, that we haven't accounted for them. Figure 15 contains the 16 unknown bytes for the four mhod blocks shown in Figure 13.

```
1)  0100 0000 3600 0000 0000 0000 0000 0000
2)  0100 0000 2e00 0000 0000 0000 0000 0000
3)  0100 0000 0a00 0000 0000 0000 0000 0000
4)  0100 0000 1e00 0000 0000 0000 0000 0000
```
Figure 15. The 16 Unknown Bytes for the Four mhod Blocks Shown in Figure 13

We find a 4-byte number that is always one, a varying 4-byte number and 64 bits of zero. On closer inspection, the second number seems to vary with the length of the subsequent string and is, in fact, its length in bytes. We will add a test to our validation program to check if this is true for all mhod strings.

Our validation program finds text data chunks where the first number is zero instead of one. These follow mhod chunks where the type is 100, probably an empty string type. Figure 16 contains a complete description of the mhod chunks.

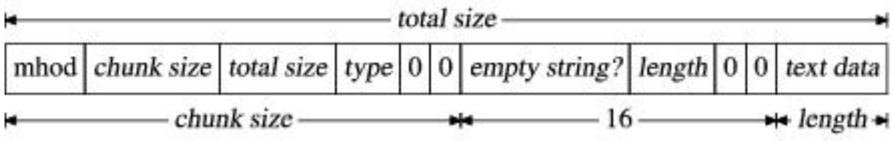


Figure 16. The Complete mhod Chunk

We can use the mhod children of the mhit chunks to figure out which songs relate to which mhit chunks. This helps us figure out the rest of the mhit chunk: n5 changes based on the file type; n7 is the size of the song's file. n8 is the duration of the song in milliseconds. n9 is the track number. n12 is the bit rate. See Figure 17 for a diagram of the mhit data chunk.

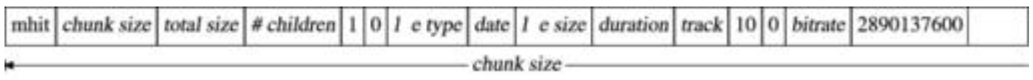


Figure 17. The mhit Data Chunk

Our analysis of the left side of the tree is complete, and using it we can find the information for all the songs on the iPod. Earlier we hypothesized that the other main use of the iTunesDB file is for playlists, so we will look for playlist information in the right side of the tree.

Hoping for symmetry, we want to find one mhyp chunk for each playlist, as there was one mhit chunk for each song. Looking around, we know this iPod has three playlists yet there are four mhyp chunks. When we run our validation program on a variety of iTunesDB files, we learn there is always one more mhyp chunk than the number of playlists.

We will use our validation program to display information about the children of the mhyp chunks. It shows us that every mhyp chunk has a first child that is an empty string mhod chunk (type 100) with a mysterious extra 604 null bytes. Then it has an mhod chunk with a string in it. For the first mhyp chunk, this is the name of the iPod; for every other mhyp chunk it is the name of a playlist.

After the name mhod chunk comes a series of mhip and mhod pairs, until the next mhyp chunk. The mhod portion of the pair is an empty string. The number of these pairs is equal to the number of songs in the playlist. As far as the first playlist is concerned, it has the same number of mhip/mhod pairs as there are songs on the iPod. Thus, we can theorize that this first playlist is a master playlist containing one entry for every song on the iPod.

We know that a playlist is a named list of songs with a title. We found the name already, but we need to figure out how the list of songs is stored. We start by looking at the mhip chunk in Figure 18. n5 is the same order of magnitude as the date we found earlier. We saw n4 (2626) in the mhit data at n2. Thus, we can guess that n4 is a song identification number and that the mhip chunks use song identification numbers to specify which songs are in the playlist. We can add a test to the validation program to match n4 in the mhip chunks, with n2 in the mhit chunks to confirm our theory.

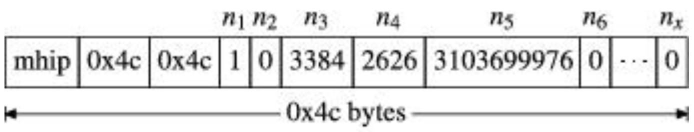

n_1 n_2 n_3 n_4 n_5 n_6 n_x
mhip | 0x4c | 0x4c | 1 | 0 | 3384 | 2626 | 3103699976 | 0 | ··· | 0
0x4c bytes

Figure 18. The mhip Chunk

The remaining mhip mystery is that the n3 number is similar in magnitude to the song key, n4. If we change our test program to look for songs whose key is the number following the key we already found, we won't find any songs with these identification numbers in any iTunesDB file. We can deduce, then, that it is a unique identification number for these chunks. Adding a test to our validation program confirms they are indeed unique. Figure 19 contains a diagram of the mhip data chunk.

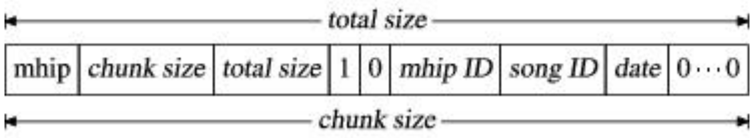

total size
mhip | chunk size | total size | 1 | 0 | mhip ID | song ID | date | 0···0
chunk size

Figure 19. The mhip Data Chunk

The mhyp chunk (see Figure 20) is the final chunk we need to evaluate. n1 is always 2. n2 is the number of songs in the playlist. The master playlist has n3 set to 1; all other playlists have it set to 0. n4 is a date, probably the date the playlist was created. n5 and n6 are the same for every mhyp chunk, and 72 null bytes always follow them.

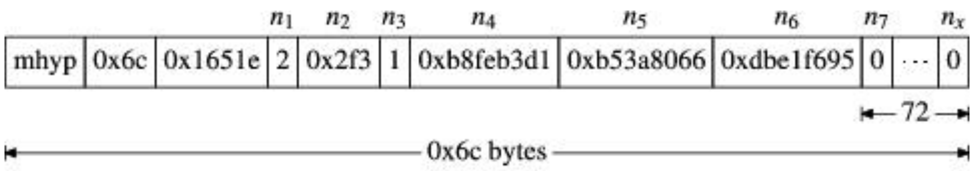

n_1 n_2 n_3 n_4 n_5 n_6 n_7 n_x
mhyp | 0x6c | 0x1651e | 2 | 0x2f3 | 1 | 0xb8feb3d1 | 0xb53a8066 | 0xdbe1f695 | 0 | ··· | 0
72
0x6c bytes

Figure 20. The mhyp Chunk

Using the three reverse engineering strategies of hypothesis, pattern recognition and validation, we deciphered the iTunesDB file format. Hopefully the ideas presented here will help Linux users reverse engineer more devices and file formats to make them compatible with Linux.

Patrick Crosby (pcrosby@tex9.com) is the founder of tex9, a Linux software company (www.tex9.com). He wrote sumi, sumipod and ipodclu. He lives and works in San Francisco, California.

Advanced search

The Serial Driver Layer

Greg Kroah-Hartman

Issue #104, December 2002

Greg explains the interface to the new serial driver layer and how to register a serival driver and an individual serial port.

In the last two installments of Driving Me Nuts [*LJ* August 2002 and October 2002], we covered the tty layer, explaining how to create a minimal tty driver. We also explained some of the various ioctls and how to interpret the termios structure. Those articles are a great start if you have to implement a new tty-type device for your embedded system, such as a serial port. For every new system that is designed, the hardware engineers always like to place the serial port at a different address, or use a different UART, or sometimes just forget the serial port and use a USB port. So most developers want to create a whole new tty driver for their new devices in order for the hardware to work properly on Linux. Fortunately, there are some layers above the tty layer that help buffer its complexities and provide the developer with a lot of common functions that are needed for a serial driver and that fit the UART or USB model better. These layers are the serial and USB to serial driver layers. We cover the serial driver layer in this article and will cover the USB to serial layer in a future article.

Serial Mess

If you look at the code for the generic PC serial driver in the 2.2 and 2.4 kernels (available at drivers/char/serial.c), you will see a very complex driver that has numerous #ifdef lines, depending on the type of hardware you are using. This file has been copied numerous times in order to provide serial support for devices that do not fit the typical PC UART device (such as the serial_amba.c and serial_21285.c drivers). Thankfully, a number of developers, led by ARM Linux maintainer Russell King, restructured the serial driver into a generic serial core and a number of smaller, hardware-specific drivers. That code showed up in the main kernel tree sometime around the 2.5.28 release. The examples in this article are from version 2.5.35, so check that things have not changed for the kernel version you are using.

Registering a Serial Driver

The serial layer requires your driver to do two things: register the driver itself with the serial core and then register the individual serial ports as they are found in the system (through PCI enumeration or some other sort of device discovery mechanism). To register your driver, call uart_register_driver() with a pointer to a struct uart_driver structure. This function takes the information provided in the uart_driver structure and sets up the tty layer based on this.

The fields in the uart_driver structure that the serial driver needs to provide are the following:

```
struct module             *owner;
const char                *driver_name;
const char                *dev_name;
int                        major;
int                        minor;
int                        nr;
struct console            *cons;
```

The owner field is a pointer to the module owning the serial driver. This is usually set to the macro THIS_MODULE.

The driver_name field is a pointer to the string that describes this driver, which is usually the same as the dev_name field. However, when describing the dev_name field, devfs needs to be taken into account, and the characters “%d” must be added to the end of this field if devfs is selected. This is because of the way devfs creates the device nodes. As an example, the amba.c driver sets the driver_name and dev_name fields as such:

```
        .driver_name              = "ttyAM",
#ifdef CONFIG_DEVFS_FS
        .dev_name                 = "ttyAM%d",
#else
        .dev_name                 = "ttyAM",
#endif
```

The major and minor fields specify the major number for the driver and the starting minor number, respectively.

The nr field specifies the maximum number of serial ports this driver supports.

The cons field is a pointer to the struct console structure that is used if this driver can support a serial console. If the driver does not support serial console mode, this field should be set to NULL.

Registering a Serial Port

Now that the serial driver has been registered with the serial driver layer, each of the serial ports needs to be registered individually with a call to uart_add_one_port(). This function takes a pointer to the original uart_driver structure that was passed to uart_register_driver and a pointer to the uart_port structure. The uart_port structure is defined as the following:

```
struct uart_port {
  spinlock_t       lock;      /* port lock */
  unsigned int     iobase;    /* in/out[bwl] */
  char             *membase;  /* read/write[bwl] */
  unsigned int     irq;       /* irq number */
  unsigned int     uartclk;   /* base uart clock */
  unsigned char    fifosize;  /* tx fifo size */
  unsigned char    x_char;    /* xon/xoff char */
  unsigned char    regshift;  /* reg offset shift */
  unsigned char    iotype;    /* io access style */
  unsigned int     read_status_mask;
                              /* driver specific */
  unsigned int     ignore_status_mask;
                              /* driver specific */
  struct uart_info *info;
                         /* pointer to parent info */
  struct uart_icount icount; /* statistics */
  struct console   *cons;
                       /* struct console, if any */
#ifdef CONFIG_SERIAL_CORE_CONSOLE
  unsigned long sysrq;       /* sysrq timeout */
#endif
  unsigned int     flags;
  unsigned int     mctrl;
                 /* current modem ctrl settings */
  unsigned int     timeout;
                 /* character-based timeout */
  unsigned int     type;      /* port type */
  struct uart_ops  *ops;
  unsigned int     line;      /* port index */
  unsigned long    mapbase;   /* for ioremap */
  unsigned char    hub6;
           /* this should be in the 8250 driver */
  unsigned char unused[3];
    };
```

A majority of these fields are used by the individual serial drivers during their operation to define how the specific port is hooked up to the processor (through the hub6, iobase, membase, mapbase and iotype variables).

One of the more interesting variables in this structure is the struct uart_ops pointer, which defines a list of functions that the serial core uses to call back into the port-specific serial driver. This structure is defined as:

```
struct uart_ops {
  unsigned int    (*tx_empty)(struct uart_port *);
  void   (*set_mctrl)(struct uart_port *,
                      unsigned int mctrl);
  unsigned int    (*get_mctrl)(struct uart_port *);
  void   (*stop_tx)(struct uart_port *,
                    unsigned int tty_stop);
  void   (*start_tx)(struct uart_port *,
                     unsigned int tty_start);
  void   (*send_xchar)(struct uart_port *, char ch);
  void   (*stop_rx)(struct uart_port *);
  void   (*enable_ms)(struct uart_port *);
  void   (*break_ctl)(struct uart_port *, int ctl);
  int    (*startup)(struct uart_port *);
  void   (*shutdown)(struct uart_port *);
  void   (*change_speed)(struct uart_port *,
                         unsigned int cflag,
                         unsigned int iflag,
                         unsigned int quot);
  void        (*pm)(struct uart_port *,
                    unsigned int state,
              unsigned int oldstate);
```

```
    int         (*set_wake)(struct uart_port *,
                            unsigned int state);
    /*
     * Return a string describing the port type
     */
    const char *(*type)(struct uart_port *);
    /*
     * Release IO and memory resources used by
     * the port. This includes iounmap if necessary.
     */
    void   (*release_port)(struct uart_port *);
    /*
     * Request IO and memory resources used by the
     * port.  This includes iomapping the port if
     * necessary.
     */
    int    (*request_port)(struct uart_port *);
    void   (*config_port)(struct uart_port *, int);
    int    (*verify_port)(struct uart_port *,
                           struct serial_struct *);
    int    (*ioctl)(struct uart_port *, unsigned int,
                    unsigned long);
  };
```

This is a very large structure, with a lot of different function pointers, looking almost as bad as the tty_driver structure.

The startup function is called once each time the open(2) call happens. It is only called after the serial core has done a lot of resource checking and is sure the port needs to be opened. The serial driver usually does some hardware-specific setup to allow the port to be used in this function.

The shutdown function is the opposite of the startup function. It is called when the port is closed and all data has stopped flowing though it. This is where the hardware is told to stop, and any resources that were allocated in the startup function should be freed.

The request_port and release_port functions are used to reserve memory and other hardware resources that are related to the serial port. The config_port function is much like request_port, but it is called when the hardware can autoprobe for any serial ports connected to it and is also responsible for doing the same hardware reservations that request_port does.

The change_speed function is called whenever the port line settings need to be modified. The values passed to this function already have been cleaned up from the original termios structure that was passed through the tty layer, making the logic in the serial driver much simpler.

There are a number of functions that are used to get and set the serial line state and port status, these are:

- set_mctrl: sets a new value for the MCR UART register.
- get_mctrl: gets the current MCR UART register value.
- stop_tx: stops the port from sending data.

- start_tx: starts the port sending data.
- tx_empty: returns if the port transmitter is empty or not.
- send_xchar: tells the port to send the XOFF character to the host.
- stop_rx: stops receiving data.
- break_ctl: sends the BREAK value over the port.
- enable_ms: enables the modem status interrupts.

There are two functions related to power management issues with the serial port: pm and set_wake. If your hardware platform supports power management, use these functions to handle powering the hardware up and down.

verify_port is called to verify that the settings passed to it are valid settings for the specific serial port and is called when the user makes a TIOCSSERIAL ioctl(2) call on the port (see “The tty Layer, Part II”, *LJ* October 2002 for more on TIOCSSERIAL).

The serial driver layer handles many of the common serial ioctls, such as TIOCMGET, TIOCMBIS, TIOCMBIC, TIOCMSET, TIOCGSERIAL, TIOCSSERIAL, TIOCSERCONFIG, TIOCSERGETLSR, TIOCMIWAIT, TIOCGICOUNT, TIOCSERGWILD and TIOCSERSWILD. If any other ioctl is called on the serial port, it is passed down to the specific port through the ioctl callback in the uart_ops structure.

The last remaining function is type, which is used to return a string describing the type of serial port. This is used in the proc file that resides in /proc/tty/driver/ and in the initial boot message when the port is discovered.

Where Is the Data?

You may have noticed there is no function in the struct uart_ops to send or receive data. The data that is sent from the user to the tty layer through a call to write(2) is placed in a circular buffer by the serial driver layer, and it is up to the specific UART driver to pick up this data and send it out of the port. Usually, every time the UART interrupt happens, the driver checks the circular buffer to see if any more data is present to be sent. The following example function shows one way of doing this:

```
static void tiny_tx_chars(struct uart_port *port)
{
    struct circ_buf *xmit = &port->info->xmit;
    int count;
    if (port->x_char) {
        /* send port->x_char out the port here */
        UART_SEND_DATA(port->x_char);
        port->icount.tx++;
        port->x_char = 0;
        return;
    }
    if (uart_circ_empty(xmit) ||
       uart_tx_stopped(port)) {
        tiny_stop_tx(port, 0);
        return;
    }
    count = port->fifosize >> 1;
    do {
        /* send xmit->buf[xmit->tail]
         * out the port here */
        UART_SEND_DATA(xmit->buf[xmit->tail]);
        xmit->tail = (xmit->tail + 1) &
                     (UART_XMIT_SIZE - 1);
        port->icount.tx++;
        if (uart_circ_empty(xmit))
            break;
    } while (--count > 0);
    if (uart_circ_chars_pending(xmit) <
       WAKEUP_CHARS)
        uart_event(port, EVT_WRITE_WAKEUP);
    if (uart_circ_empty(xmit))
        tiny_stop_tx(port, 0);
}
```

The function starts out by looking to see if the x_char is specified to be sent out at this moment in time. If so, it sends it out and increments the number of characters sent out the port. If not, the circular buffer is checked to see if it has any data in it and if the port is not currently stopped by anything. If this is true, we start taking characters out of the circular buffer and send them to the UART. In this example, we send, at most, the size of the FIFO divided by two, which is a good average amount of characters to send. After the characters are sent, we see whether we have flushed out enough characters from the circular buffer to ask for more to be sent to us. If so, we call uart_event() with the EVT_WRITE_WAKEUP parameter, telling the serial core to notify user space that we can accept more data.

Any data received by the serial driver is passed to the tty layer, exactly like a normal tty driver would, with a call to tty_insert_flip_char(). This is usually done in the UART interrupt function.

Tiny Serial Example

Listing 1 [available on the *LJ* FTP site at ftp.linuxjournal.com/pub/lj/listings/issue104/6331l1.txt], shows an example of how to register a serial driver with the serial driver layer and how to register a single serial port. This serial port is much like the previous tty example serial driver in that it acts like a character is received every two seconds as long as the port is opened. It also will print any characters that are sent to it with a call to write(2) to the kernel debug log.

Conclusion

In this article, the interface to the new serial driver layer has been explained, detailing how to register a serial driver and then an individual serial port. Thanks to this new driver layer being introduced in the 2.5 kernel, serial drivers can be much smaller, and creating a new one is a easier process, keeping the complexities of the tty layer away from the programmer.

Greg Kroah-Hartman is currently the Linux USB and PCI Hot Plug kernel maintainer. He works for IBM, doing various Linux kernel-related things and can be reached at greg@kroah.com.

Archive Index Issue Table of Contents

Advanced search

Trees in the Reiser4 Filesystem, Part I

Hans Reiser

Issue #104, December 2002

The basic structure of the new ReiserFS—graphs vs. trees, keys, nodes and blocks.

One way of organizing information is to put it into trees. When we organize information in a computer, we typically sort it into piles called nodes, with a name (pointer) for each pile. Some of the nodes contain pointers, and we can look through the nodes to find those pointers to (usually other) nodes.

We are interested particularly in how to organize so we actually can find things when we search for them. A tree is an organizational structure that has some useful properties for that purpose. We define a tree as follows:

1. A tree is a set of nodes organized into a root node and zero or more additional sets of nodes, called subtrees.
2. Each of the subtrees is a tree.
3. No node in the tree points to the root node, and exactly one pointer from a node in the tree points to each non-root node in the tree.
4. The root node has a pointer to each of its subtrees, that is, a pointer to the root node of the subtree.

Note that a single node with no pointers is a tree, because it is the root node. Also, a tree can be a linear tree of nodes without branches.

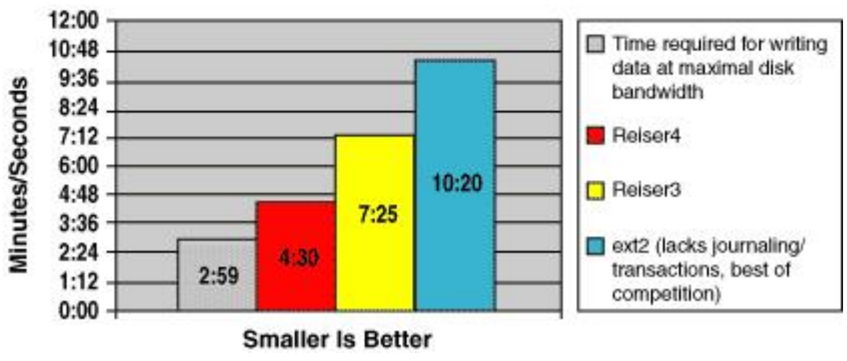


Figure 1. Write Performance (30 Copies Linux Kernel 2.4.18)

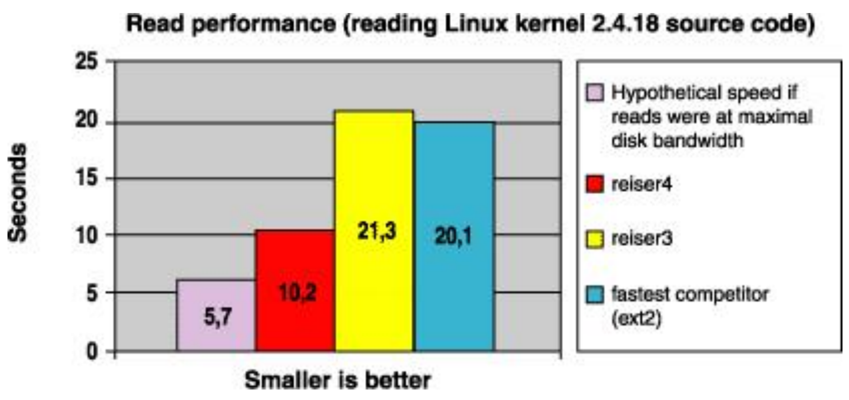


Figure 2. Read Performance (Reading Linux Kernel 2.4.18 Source Code)

Fine Points of the Definition

It is interesting to argue about whether finite should be a part of the definition of trees. There are many ways of defining trees, and which is the best definition depends on your purpose. Professor Donald Knuth supplies several definitions of tree. As his primary definition of tree he even supplies one which has no pointers, edges or lines in the definition, only sets of nodes.

Knuth defines trees as being finite sets of nodes. There are papers on inifinte trees on the Internet. I think it more appropriate to consider finite an additional qualifier on trees, rather than bundling finite into the definition. However, I personally only deal with finite trees in my research.

Edge is a term often used in tree definitions. A pointer is unidirectional, meaning you can follow it from the node that has it to the node it points to, but you cannot follow it back from the node it points to, to the node that has it. An edge, however, is bidirectional, meaning you can follow it in both directions.

Here are three alternative tree definitions, which are interesting in how they are mathematically equivalent to each other. They are not equivalent to the definition I supplied, because edges are not equivalent to pointers. For all three of these definitions, let there be no more than one edge connecting the same two nodes:

- a set of vertices (aka points) connected by edges (aka lines) for which the number of edges is one less than the number of vertices;
- a set of vertices connected by edges that have no cycles (a cycle is a path from a vertex to itself);
- or a set of vertices connected by edges for which there is exactly one path connecting any two vertices.

These three alternative definitions do not have a unique root in their tree, and such trees are called free trees.

The definition I supplied is a definition of a rooted tree, not a free tree. It also has no cycles, it has one less pointer than it has nodes, and there is exactly one path from the root to any node.

Graphs vs. Trees

Consider the purposes for which you might want to use a graph and those for which you might want to use a tree. In a tree there is exactly one path from the root to each node in the tree, and a tree has the minimum number of pointers sufficient to connect all the nodes. This makes it a simple and efficient structure. Trees are useful when efficiency with minimal complexity is desired and when there is no need to reach a node by more than one route.

Reiser4 has both graphs and trees, with trees used when the filesystem chooses the organization (in what we call the storage layer, which tries to be simple and efficient) and graphs when the user chooses the organization (in the semantic layer, which tries to be expressive so that the user can do whatever he or she wants).

Keys

We assign a key to everything stored in the tree. We find things by their keys, and using them gives us additional flexibility in how we sort things. If the keys are small, we have a compact means of specifying enough information to find the thing. It also limits what information we can use for finding things.

This limit restricts the key's usefulness, and so we have a storage layer, which finds things by keys, and a semantic layer, which has a rich naming system (described in Part II of this article). The storage layer chooses keys for things solely to organize storage in a way that improves performance, and the semantic layer understands names that have meaning to users. As you read, you might want to think about whether this is a useful separation that allows the freedom to add improvements that aid performance in the storage layer, while escaping paying a price for the side effects of those improvements on the flexible naming objectives of the semantic layer.

Choosing Which Subtree

We start our search at the root, because from the root we can reach every other node. But, how do we choose which subtree of the root to go to from the root? The root contains pointers to its subtrees. For each pointer to a subtree there is a corresponding left delimiting key. Pointers to subtrees, and the subtrees themselves, are ordered by their left delimiting key. A subtree pointer's left delimiting key is equal to the least key of the things in the subtree. Its right delimiting key is larger than the largest key in the subtree, and it is the left delimiting key of the next subtree of this node.

Each subtree contains only things whose keys are at least equal to the left delimiting key of its pointer and are not more than its right delimiting key. If there are no duplicate keys in the tree, then each subtree contains only things whose keys are less than its right delimiting key. If there are no duplicate keys, then by looking within a node at its pointers to subtrees and their delimiting keys, we know which subtree of that node contains the thing we want.

Duplicate keys are a topic for another time. For now I will only hint that when searching through objects with duplicate keys we find the first of them in the tree. Then we search through all the duplicates, one by one, until we find what we are looking for. Allowing duplicate keys can allow for smaller keys, so there is sometimes a trade-off between key size and the average frequency of such inefficient linear searches.

The contents of each node in the tree are sorted within the node. Therefore, the entire tree is sorted by key, and for a given key we know exactly where to go to find at least one thing with that key.

Node Size

We choose to make the nodes equal in size, so it is easier to allocate the unused space between nodes, because it will be of a size equal to some multiple of the size of a node. This way, there aren't any problems of space being free but not large enough to store a node. Also, disk drives have an interface that assumes equal size blocks, which they find convenient for their error-correction algorithms.

If having the nodes be equal in size is not very important, perhaps because the tree fits into RAM, then using a class of algorithms called skip lists is worth considering.

Reiser4 nodes are usually equal to the size of a page, which if you use Linux on an Intel CPU is 4,096 (4k) bytes. There is no measured empirical reason to think this size is better than others; it is simply the one that Linux makes easiest and cleanest to program. Quite honestly, we have been too busy to experiment with other sizes.

Sharing Blocks Saves Space

If nodes are of equal size, how do we store large objects? We chop them into pieces, and we call these pieces items. An item is sized to fit within a single node.

Conventional filesystems store files in whole blocks. Roughly speaking, this means that, on average, half a block of space is wasted per file because not all of the last block of the file is used. If a file is much smaller than a block, the space wasted is a large percentage of the total file size. It is not effective to store such typical database objects as addresses and phone numbers in separately named files in a conventional filesystem because more than 90% of the space in the storage blocks is wasted. By putting multiple items within a single node in Reiser4, we are able to pack multiple small pieces of files into one block. Our space efficiency is roughly 94% for small files. This does not count per-item-formatting overhead, whose percentage of total space consumed depends on average item size and, for that reason, is hard to quantify.

Aligning files to 4k boundaries does have advantages for large files though. When a program wants to operate directly on file data without going through system calls to do it, it can use mmap() to make the file data part of the process' directly accessible address space. Due to some implementation details, mmap() needs file data to be 4k-aligned. If the data is already 4k-aligned, it makes mmap() much more efficient. In Reiser4 the current default is that files larger than 16k are 4k-aligned. We don't yet have enough empirical data and experience to know whether 16k is the optimal default value for this cutoff point.

Leaves, Twigs and Branches

Continuing with our tree metaphor, leaves are nodes that have no children. Internal nodes are nodes that have children.

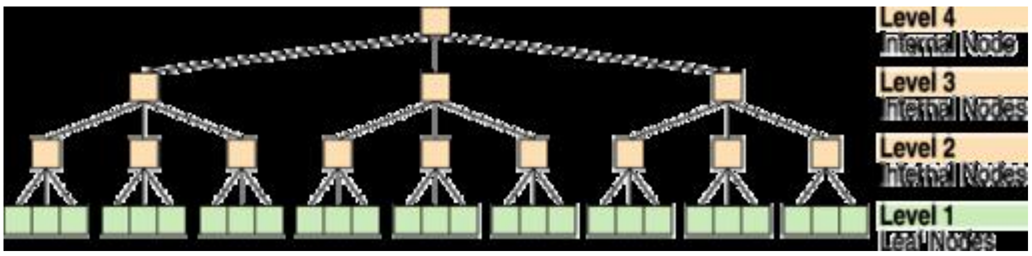


Figure 3. A Height = 4, Four-Level, Fanout = 3, Balanced Tree

A search starts with the root node, traverses two more internal nodes and ends with a leaf node, which is a node that holds the data and has no children.

An item is a data container that is contained entirely within a single node. A node that contains items is called a formatted node. When we store objects in the tree, we put their various parts into items and unformatted leaves. Unformatted leaves (unfleaves) are leaves that contain only data and do not contain any formatting information. Only leaves can contain unformatted data. Pointers are stored in items, and so all internal nodes are necessarily formatted nodes.

Pointers to unfleaves are different in their structure from pointers to formatted nodes. To clarify, extent pointers point to unfleaves. An extent is a sequence of contiguous, in block number order, unfleaves that belong to the same object. An extent pointer contains the starting block number of the extent and a length. Because the extent belongs to only one object, we can store only one key for the extent, and we can calculate the key of any byte within that extent. If the extent is at least two blocks long, extent pointers are more compact than regular node pointers.

Node pointers are pointers to formatted nodes. We do not yet have a compressed version of node pointers, but they should be available soon. Notice that with extent pointers we don't have to store the delimiting key of each node pointed to, while we need to do this when using node pointers. We will probably introduce key compression at the same time we add compressed node pointers. One would expect keys to compress well because they are sorted into ascending order. We expect our node and item plugin infrastructure will make such features easy to add at a later date.

Twigs are parents of leaves, and extent pointers exist only in twigs. (This is a very controversial design decision I will discuss in Part II of this article.) Branches are internal nodes that are not twigs.

You might think we would number the root level 1, but since the tree grows at the top, it turns out to be more useful for number 1 to be the level with the leaves where object data is stored. The height of the tree depends upon how many objects we have to store and what the fanout rate (average number of children) of the internal and twig nodes will be.

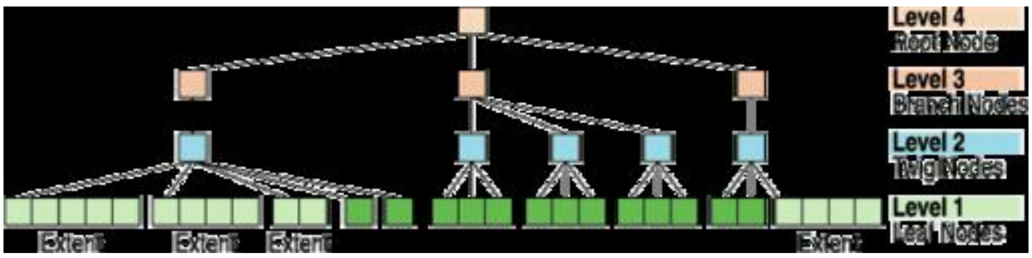


Figure 4. A Reiser4 Tree, Including the Internal Nodes Called Twig Nodes

Types of Items

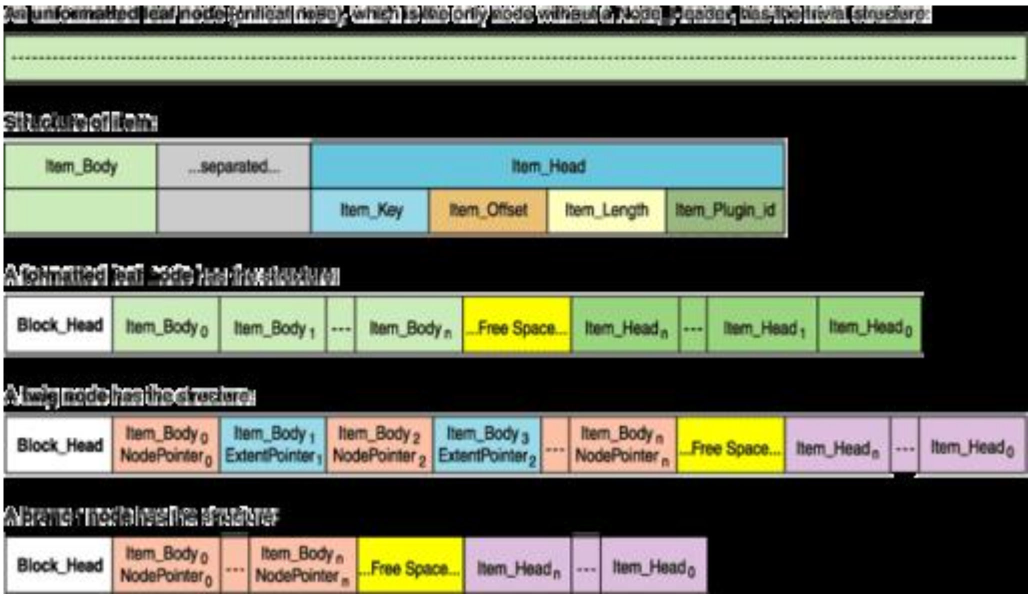
Reiser4 includes many different kinds of items designed to hold different types of information:

- static_stat_data: holds the owner, permissions, last access time, creation time, last modification time, size and the number of links (names) to the file.
- cmpnd_dir_item: holds directory entries and the keys of the files they link to.
- extent pointers: explained above.
- node pointers: explained above.
- bodies: hold parts of files not large enough to be stored in unfleaves.

Units

We call a unit that which we must place as a whole into an item, without splitting it across multiple items. When traversing an item's contents, it is often convenient to do so in units:

- For body items the units are bytes.
- For directory items the units are directory entries. The directory entries contain a name and a key of the file named (in practice the name and key may be compressed).
- For extent items the units are extents. Extent items contain only extents from the same file.
- For static_stat_data the whole stat data item is one indivisible unit of fixed size.


An unformatted leaf node (unfleaf node), which is the only node without a Node_Header, has the trivial structure:
Structure of Item:
Item_Body
...separated...
Item_Head
Item_Key
Item_Offset
Item_Length
Item_Plugin_id
A formatted leaf node has the structure:
Block_Head
Item_Body 0
Item_Body 1

Item_Body n
...Free Space...
Item_Head n

Item_Head 1
Item_Head 0
A twig node has the structure:
Block_Head
Item_Body 0 NodePointer 0
Item_Body 1 ExtentPointer 1
Item_Body 2 NodePointer 2
Item_Body 3 ExtentPointer 2

Item_Body n NodePointer n
...Free Space...
Item_Head n

Item_Head 0
A branch node has the structure:
Block_Head
Item_Body 0 NodePointer 0

Item_Body n NodePointer n
...Free Space...
Item_Head n

Item_Head 0
Figure 5. What Node Formats Look Like

Conclusion

I have explained the basic structures of the Reiser4 tree, but the fun stuff is yet to come. I have not yet explained how other researchers structure their trees. Nor did you learn why object contents are stored at the bottom of the tree, why high fanout is important or what are the different kinds of balancing. No hint have I yet given as to why balanced trees are better and dancing trees are best. What I have most especially not done is explain how a subtle and controversial tree structure change, which you can see in the trees depicted in this article, doubled Reiser4 read speed compared to Reiser3. This will (space permitting) be in Part II in next month's issue of *Linux Journal*.

Resources

Hans Reiser (reiser@namesys.com) entered UC Berkeley in 1979 after completing the eigth grade and majored in “Systematizing”, an individual major based on the study of how theoretical models are developed. His senior thesis discussed how the philosophy of the hard sciences differs from that of computer science, with the development of a naming system as a case study. He is still implementing that naming system, of which Reiser4 is the storage layer. In 1993 he went to Russia and hired a team of programmers to develop ReiserFS. He worked full-time to pay their salaries while spending nights and weekends arguing over algorithms. In 1999 it began to work well enough that his mother stopped suggesting a salaried job at a nice big company.

Creating OpenACS Packages

Reuven M. Lerner

Issue #104, December 2002

Reuven shows how to develop your own simple web/database application using APM.

Last month, we continued our exploration of OpenACS by looking at how individual applications are packaged together using APM (the ArsDigita Package Manager). Every OpenACS application normally combines database tables with server-side programs, which means that installing or upgrading a package is more complicated than copying some files. As we saw last month, the APM application makes it relatively easy to install a package and then instantiate it under one or more URLs.

This is great if you care only about using existing APMs. But most OpenACS installations will require new, custom packages that have their own data models and programs. While it's theoretically possible to develop OpenACS applications without APMs, doing so makes it hard to distribute your software, track versions or standardize your installation procedure.

This month, we cover how to develop our own simple web/database application using APM. The final result is an application that someone else can load into their OpenACS system.

Creating the Skeleton Package

The first step in creating a new OpenACS application is making a new skeleton package via the web-based APM program. By default, this program is available only to the user with administration rights to an OpenACS system, so you might need to ask the administrator to modify the permissions when you begin your development work.

On most OpenACS systems, you can launch the APM program via the /acs-admin/apm/ URL. This program displays all of the installed APMs, including each one's name, version and number of files it includes. The file count normally includes .sql files (for creating and removing database definitions), .tcl files (containing Tcl program code), .adp files (for web templates similar to ASP or JSP) and .xql files (containing SQL queries). However, an APM can contain other files, including images, text files or even something more unusual, such as Flash.

As we saw last month, we can use this screen as an entry point to install, inspect and modify the packages on our system. But if we go down to the bottom of this list and click the create a new package link, we can begin the process of creating our own application.

The initial “create a package” screen asks for a number of parameters that will help APM create a .info file that describes the package to the system. Assuming that we want to create a “hello, world” application with a minimum amount of work, we can fill in a small number of fields:

- The package key should be atf-hello. There is no namespace hierarchy for APMs, but most developers put their name (or a similar identifying term) at the front of the package name to avoid conflicts.
- The package name can be Hello or anything else you choose; it is used by developers to distinguish it from other packages.
- We are developing an application, not a service.
- Our version number is 0.1d, meaning that it's in early stages of development.
- You can fill out or ignore the summary and description at your discretion. It's a good idea to fill these in for actual applications, however.

When finished, ensure that the “write a package” specifier box is checked, and click the “create package” button at the bottom of the page. You will be taken to the management screen for this particular package. If you look in the packages subdirectory on the filesystem where your OpenACS toolkit was installed, you'll see an atf-hello directory, containing an atf-hello.info file in the correct XML format.

Create a Data Model

Now that OpenACS recognizes our package, we can get to work designing the data model for our package. As all experienced web/database developers know, designing the tables is the hard part; once you know how they will work, creating the applications that add, delete and modify that data is pretty easy.

Our application will give us a simple guest book, in which visitors to the site can enter comments on our page. (OpenACS comes with a more general and powerful facility for doing this on any page on the site, but we will ignore this in the interest of learning how to create a package.) Our data model will look like this:

```
CREATE TABLE atf_hello_postings (
  posting_id    SERIAL   NOT NULL,
  user_id       INTEGER  NOT NULL  REFERENCES users
                ON DELETE CASCADE
                ON UPDATE CASCADE,
  entry_date    TIMESTAMP  NOT NULL  DEFAULT NOW(),
  posting       TEXT     NOT NULL
                CHECK (posting <> ''),
  PRIMARY KEY(posting_id)
);
```

On a purely technical level, the previous table definition probably looks reasonable to anyone who has worked with PostgreSQL before. We set up posting_id as an integer primary key, user_id as a foreign key, a timestamp field (containing date and time information) and then a text field to contain the actual posting. Notice how our table name begins with atf_hello, reflecting the fact that it is in the atf-hello APM. Keeping table names consistent with package names is a primitive form of namespace management, but it works well enough if everyone sticks with it.

The above works under PostgreSQL but isn't guaranteed to work with Oracle. Given that the OpenACS community prides itself on working transparently with both databases, what should we do to avoid alienating our high-end colleagues?

The answer is that when installing the atf-hello package, APM looks for a file named sql/atf-hello-create.sql. If such a file exists, it is assumed to work on all support databases. If not, APM looks for subdirectories named postgresql and oracle and executes atf-hello-create.sql in the appropriate directory. So if your system uses PostgreSQL, the above SQL would be saved in a file named sql/postgresql/atf-hello-create.sql. Official OpenACS packages are supposed to work with both Oracle and PostgreSQL out of the box, which means that it's rare to find a package that works with only one brand or the other. (The examples in this month's column are guaranteed to work only with PostgreSQL, although porting them to Oracle should not be too difficult.)

OpenACS also allows us to create a cleanup script, named sql/PACKAGE-drop.sql, that removes all the tables and stored procedures that the create script defined. We thus create a file named sql/postgresql/atf-hello-drop.sql.

APM knows how to create the data model when a package is first installed, but not when you're doing development work. So to install atf-hello into the database, you'll need to handle it manually:

```
psql -f atf-hello-create.sql openacs4
```

Of course, this assumes that openacs4 is the name of your OpenACS database, that the PostgreSQL server is running on the same machine as the psql client and that your current user name has access rights to that database.

OpenACS Templating

Now that our data model is installed, it's time to write an application that uses it. OpenACS 4 introduced a new templating system that builds upon ADP (the AOLserver equivalent of ASP or JSP), which I have found to be one of the best parts of OpenACS.

ASP-like pages are easier for nonprogrammers to understand than standard server-side programs. However, hybrids of HTML and programs tend to become bottlenecks, because the designer and developer cannot work on the file simultaneously.

OpenACS templates are a refreshing solution to this problem. We divide the page into two parts: one (the .tcl page) for the programmer and the other (the .adp page) for the designer. The .tcl page begins with a contract describing which values it expects to receive, as well as which values it will provide to the ADP page. The Tcl page ends with a call to ad_return_template, which looks for an .adp page of the same name, substitutes variables appropriately and then runs the ADP parser on the page.

The Tcl page can pass values to the ADP page in the form of data sources, a fancy term for variable. If the Tcl page says

```
set five 5
```

the ADP page can retrieve this value anywhere by surrounding the variable name with @ signs, as in @five@. If the variable five has not been defined or exported in the page contract, OpenACS produces a runtime error, complaining that no such variable exists.

This splitting of responsibilities means that the designer and programmer can work independently, so long as the agreed-to interface described in the page contract (for Tcl page inputs and outputs) remains intact.

Create the Tcl Pages

OpenACS comes with a large number of functions designed to help programmers create HTML forms quickly and easily. For the purposes of this introduction, we won't use these functions; we will use simple, raw HTML instead.

Our application will consist of two URLs:

- One will display the current entries in our atf_hello_postings table, in chronological order, along with a form for entering a new posting. We will use the OpenACS database API, which is easier to use than the native AOLserver database API (and frees us from having to worry about threads and database pooling), to retrieve results from the database and stick them into a multirow variable. This variable then will be passed as a data source to the ADP page, where it will be displayed. The action for this form will point to a second URL.
- The other is the Tcl program that receives the HTML form, enters a new row into the database and redirects people back to the first page.

In other words, we will need two Tcl pages and one ADP page for our application. These will all go in the www subdirectory under atf-hello; if that directory doesn't exist, create it, double-checking that it is owned by the same user as AOLserver is running.

The first Tcl page (posting.tcl), shown in Listing 1, expects no parameters and exports a single data source named **postings**. The Tcl page begins with a call to ad_page_contract, which allows us to comment on the owner and purpose of the file (in the first parameter), the inputs we receive (in the second parameter, blank posting.tcl) and the data sources we export (in the third parameter, named properties for historical reasons). The Tcl page ends with a call to ad_return_template, which looks for an .adp file with the same name as the current .tcl file.

Listing 1. posting.tcl

While each data source is really a simple Tcl variable, the OpenACS templating system disguises the true nature of these variables somewhat, assigning each data source a name and a type (multirow, list, onevalue or onerow). In this particular case, our postings data source is a multirow, which means it will contain multiple result rows from our SELECT. The db_multirow procedure takes three arguments: the name of the variable into which the rows should be read, the name of the query and the SQL itself.

Named queries are both a blessing and a curse in OpenACS. They are a blessing, in that they make it possible to work with multiple databases by placing the query in an external .xql file (XML-formatted files appropriate for particular databases). The problem is that OpenACS first looks for an .xql file associated with the query name, only looking at the SQL in your .tcl page if no XML exists. Many beginning OpenACS programmers are surprised to find that the system is ignoring their SQL modifications in the .tcl page, looking instead at the .xql page.

Create the ADP Page

When posting.tcl ends by invoking ad_return_template, the OpenACS templating system looks for posting.adp. Because our Tcl page contained all of the Tcl program code, we don't need the standard <% %> tags in our ADP page. However, we do need a way to translate our data source into something usable within HTML.

Listing 2. posting.adp

Shown in Listing 2 (posting.adp), the OpenACS templating system adds a few tags that make it possible to retrieve the values set in the .tcl page in a straightforward way:

- We can include HTML conditionally in the output page by using an <if> tag, which compares (using eq and ne) two values. In posting.adp, we compare the number of rows in the postings data source (by querying @postings:rowcount@) with 0. If there are no rows to display, then we show nothing at all.
- If there are rows to display, we iterate through them with the <multiple> tag. Within the <multiple> tag block, we can retrieve individual database columns with the @NAME.column@ syntax, as you can see in posting.adp.

Regardless of how many rows we display, posting.adp always includes a small HTML form that sends its contents to the posting-add program. This program, posting-add.tcl (shown in Listing 3), starts with an ad_page_contract that declares a single input variable (posting_text). Input variables are mandatory by default, although we can declare them as optional or assign a default value by modifying the entry in ad_page_contract. In this particular case, we ask OpenACS to remove any leading or trailing whitespace from the text input we receive.

Listing 3. posting-add.tcl

We then retrieve the current user ID using the built-in ad_get_user_id procedure, assigning that to the user_id variable. Next, we use db_dml to insert the posting into the database. Notice how we use colons (rather than dollar signs) before the variable names in db_dml; this is standard in the OpenACS database API and ensures that we will not encounter quoting problems when passing data to the database server.

Finally, posting-add.tcl ends by redirecting the user to posting, which invokes posting.tcl and displays posting.adp.

Finishing the Package

We can now return to APM and generate a package with our templates and database creation scripts. Click on the atf-hello package name, and then click on the manage file information link toward the bottom of the page. Now click on scan for additional files in this package. You should see a list of the .sql, .tcl and .adp pages we installed. Indicate that all of these files should be included in the package, and after returning to the main ATF Hello APM management screen, click on the generate a new atf-hello.info file link.

You're now set to create an APM that can be distributed to any other OpenACS user. Click on the generate a file link, and the distribution file information will indicate the size of the generated APM. If you click on this link, an APM should be downloaded to your system.

How do you install a new APM someone has sent you? The easiest method is to place the APM on the server filesystem. Then from within your web browser, return to the main APM page (/acs-admin/apm/) and click on the install link. Tell the system where the APM is located, and it will be placed under the packages directory. You will then be able to install it using the APM installer that we examined last month. The data model will be inserted into the database, and the web pages will be made available for any interested parties. And of course, once a package is installed in the system, you can use the ACS site map application to mount a new instance of the package under a URL of your choice.

What Are We Missing?

This example package only scratches the surface of OpenACS application development, for example:

- The templating system comes with an automatic form-builder system that makes it easy to create HTML forms that automatically provide confirmation screens and data validation.
- We can load Tcl procedures into AOLserver at startup time by defining them within the package's tcl directory.
- Named SQL queries, as mentioned above, make it possible to write a single Tcl program that transparently accesses both Oracle and PostgreSQL.
- Each instance of a package can be kept separate from its peers using the OpenACS concept of context.

- Each instance can set its own parameters, allowing it to have installation-specific information.
- Each package can define (or use) its own set of permissions, allowing you to create custom permissions and custom access control lists for users and groups on the system.

Conclusion

OpenACS is complex, and APM is not the simplest system to learn because it tries to handle so many complicated cases that web/database developers often encounter. At the same time, I haven't yet seen an easier way to distribute web/database applications with this degree of modularity, portability across databases and flexibility when it comes to the templates. The ease of creating such applications, combined with a rich data model and a large set of established applications makes OpenACS a viable and useful platform for on-line communities.

email: reuven@lerner.co.il

Reuven M. Lerner is a consultant specializing in web/database applications and open-source software. His book, Core Perl, was published in January 2002 by Prentice Hall. Reuven lives in Modi'in, Israel, with his wife and daughter.

A Process Smorgasbord

Marcel Gagné

Issue #104, December 2002

Marcel serves up some truly mind-expanding ways to monitor your processes.

François, *vite*! Our guests will be here any moment. *Quoi*? You say you already have prepared everything? Excellent, François! I see you have brought up a healthy supply of the 1998 Barossa Valley Shiraz. It will pair nicely with tonight's menu, don't you think? *Qu'avez-vous dit?* Ah, the theme of this issue...it is System Administration, *mon ami*, and tonight I have decided to look into process management. But of course, François, even something as basic as processes can be the center of exciting dishes.

What did you say, François? Ah, our guests are here! Welcome, *mes amis*, to *Chez Marcel*, home of fine Linux cooking, tantalizing atmosphere and incredible wine. Your tables and the wine are ready. Please sit and my faithful waiter will fill your glasses. I must tell you that François is being unusually efficient tonight. I don't know what has gotten into him. One less thing to manage, *non*?

As everyone in this restaurant knows, everything running on your Linux machine is a process; every shell, every open connection to the Internet, every game—everything. In some cases, programs will spawn multiple processes of their own. These are child processes. Technically, every process is a child process of some parent except one—that is, init, the master process. Children can spawn more children, and they can spawn more. You can use **ps** to list them all, but in time, monitoring all this procreation can be quite exhausting. *Mon Dieu, mes amis*, I think I need a sip of my wine right now.

To obtain a quick-and-dirty view of what process is the child of what other process, you can type the **pstree** command. Note the first few lines of output below and init's position:

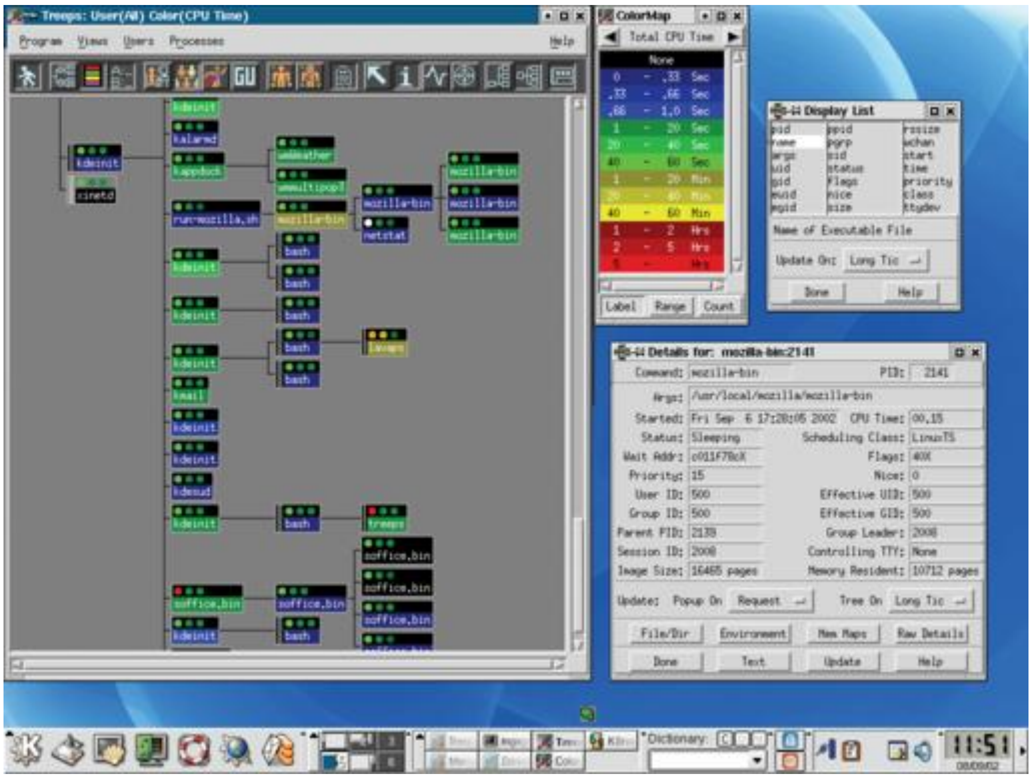
```
init-+-apmd
     |-atd
```

```
        |-bdflush
        |-cardmgr
        |-crond
        |-gpm
        |-kalarmd
        |-kapmd
        |-kappdock-+-wmWeather
        |          `-wmmultipop3
        |-kdeinit-+-artsd
        |         |-autorun
        |         |-kdeinit
        |         |-kdeinit---2*[bash]
        |         |-kdeinit---bash
        |         |-kdeinit-+-bash---lavaps
```

While this looks neat, it is somewhat lacking in information. You can get the same effect (but with more information) by using your old friend, the **ps** command. The f option displays a *forest* view through which you can see the process trees. A little joke, *mes amis*.

Transforming the sea of processes into something that readily catches the eye is exactly what George MacDonald had in mind when he created Treeps. This program is an interactive, graphical process monitor with an on-the-fly, color-coded display, thereby making it easier to nail down individual tasks. This one is certainly worth the download, *mes amis*. Get the source at the following URL: www.orbit2orbit.com/gmd/tps/treepsfm.html.

To build Treeps, extract the source (with **tar -xzvf treeps-1.2.1.tar.gz**, then run the **./Setup** script from the installation directory. After the prebuild configuration takes place, you'll be told to do a **make install**. You then run the program by typing **treeps &**.


Treeps: User(All) Color(CPU Time)
Program Views Users Processes Help
ColorMap
Total CPU Time
Display List
Label Range Count
Done Help
Details for: mozilla-bin:2141
File/Dir Environment Mem Maps Raw Details
Done Text Update Help
11:51

```
Command: mozilla-bin                              PID: 2141
   Args: /usr/local/mozilla/mozilla-bin
Started: Fri Sep  6 17:28:05 2002       CPU Time: 00.15
 Status: Sleeping               Scheduling Class: LinuxTS
Wait Addr: c011F78cX                       Flags: 40X
Priority: 15                                Nice: 0
 User ID: 500                      Effective UID: 500
Group ID: 500                      Effective GID: 500
Parent PID: 2139                    Group Leader: 2008
Session ID: 2008                 Controlling TTY: None
Image Size: 16465 pages        Memory Resident: 10712 pages
Update:  Popup On  Request        Tree On  Long Tic
```

Figure 1. Process Nirvana, Courtesy of Treeps

The initial view is of your own processes as launched from init, and this is where the fun starts. Moving your mouse pointer over a process displays some basic information, the equivalent of what you would get from a **ps x**. Right-click and a pop-up menu will give you the options of renicing the process, viewing its man page and so on.

From the bar of buttons on the top, click the various selections. Aside from a view of your own processes, you can choose to see the dæmon processes or simply *everything*. If you click the information button (the one with an “i” on it), your mouse pointer changes to an “i” as well. Click any process, and an information window appears with more details about the running process than you would have thought were there. From that window, you can drill down even further. Click the File/Dir button, and you can see every file open by that process. For the truly curious, the Mem maps button displays where in memory every chunk of code resides.

The features are numerous to say the least, but the color-coding is what really caught my eye. While the program is running, turn on the “color map viewer” by clicking on the color-bar button. By cycling through the various options, you can highlight processes based on user ID, group ID, total CPU time, current CPU load, process status (sleeping, running, zombie, etc.), resident memory, image size and more.

Under the Program menu, there's another little treat called System Info, which brings up the System Information App Launcher. From this button-laden window, you quickly can view tons of information about your system, from your routing table to loaded modules, kernel level, PCI devices, uptime, disk partitions, runlevels and so on.

Viewing things from a different perspective can give you a new appreciation for even the familiar. Indeed, it can be a mind-expanding experience, *non*? As whimsical as the next item on our menu appears to be, I found it a lot of fun to watch and work with. Whether it makes a great process monitor or not will depend on your feelings toward lava lamps. Written by John Heidemann, LavaPS was inspired by the idea of calm computing from “The Coming Age of Calm Technology” by Mark Weiser and John Seely Brown.

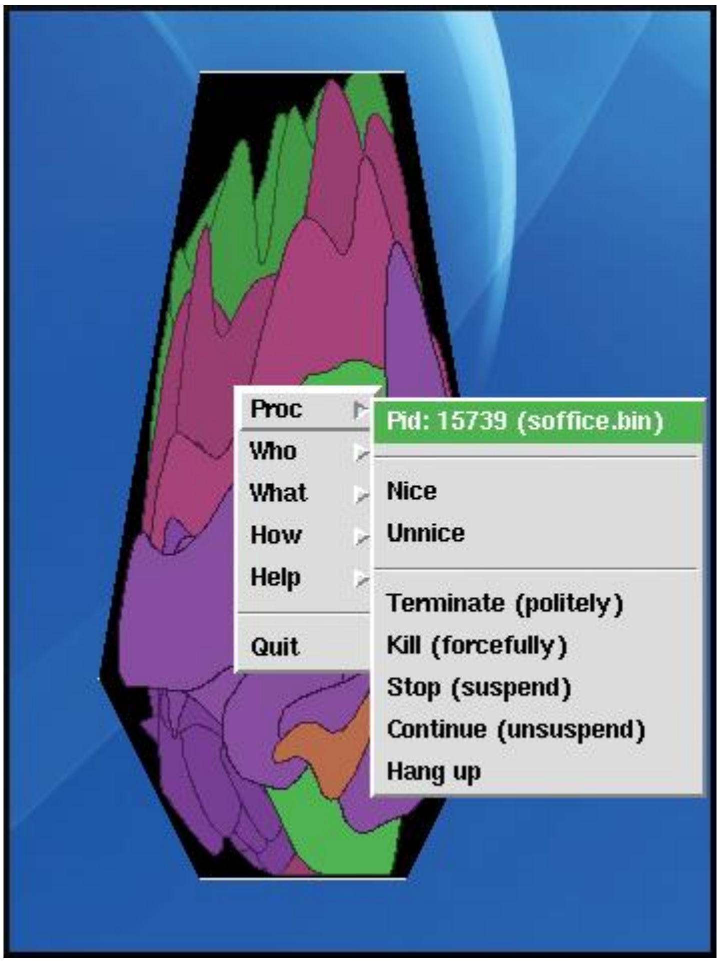


Figure 2. Mind-Expanding Process Management, Lava-Lamp Style

The idea here is that processes are represented as fluid blobs in a lava lamp. The larger the blob, the greater its memory usage. The faster it moves, the greater its CPU usage. Like any decent process monitor, it allows for identification of processes, renicing and killing. Start by visiting the lavaps web site (www.isi.edu/~johnh/SOFTWARE/LAVAPS/index.html) and picking up a copy of the package.

For the Red Hat users out there, prepackaged RPMs are available. For the others, never fear—building LavaPS is simply another example of the famous (dare I say “classic”) extract and build five-step:

```
tar -xzvf lavaps-1.20.tar.gz
cd lavaps-1.20
./configure
make
su -c make install
```

To start your lava lamp, type **lavaps &**. You'll see a small lava lamp appear on your desktop. Right-click, and a menu pops up offering a number of options. The proc menu tells you the process ID and the name of the process. It also allows you to send various signals (such as kill) to the process, from forceful termination to temporary suspension.

Running LavaPS to monitor and administer processes certainly sets an otherworldly kind of mood. The one thing I did not like is that the default lamp was actually fairly small on my 1024 × 768 display. Overriding this requires that you set X resources. This is done easily by modifying the $HOME/.lavapsrc configuration file. In mine, the only thing I changed was the geometry. Here's what my .lavapsrc file looks like:

```
lavaps.geometry: 204x404+700+0
```

Speaking of otherworldly, *mes amis*, the strangest excursion into the secret life of your processes is probably highlighted by an unusual game of *Doom*, the classic 3-D shooter from ID Software. Back in 1997, ID Software released the source code to *Doom*, and many ports followed. One of them was *XDoom*, a UNIX X Window System version on which David Koppenhofer's psDooM is based. As psDooM was inspired by *XDoom*, David was inspired by Dennis Chao, and Dennis was inspired by Vernor Vinge. If you are curious, check out the link to Dennis' “Doom as a tool for system administration” (see Resources).

Anyhow, the idea behind psDooM is to provide a strange alternative to process management. The monsters roaming the halls have red process IDs floating above their heads along with the last seven characters of the command name.

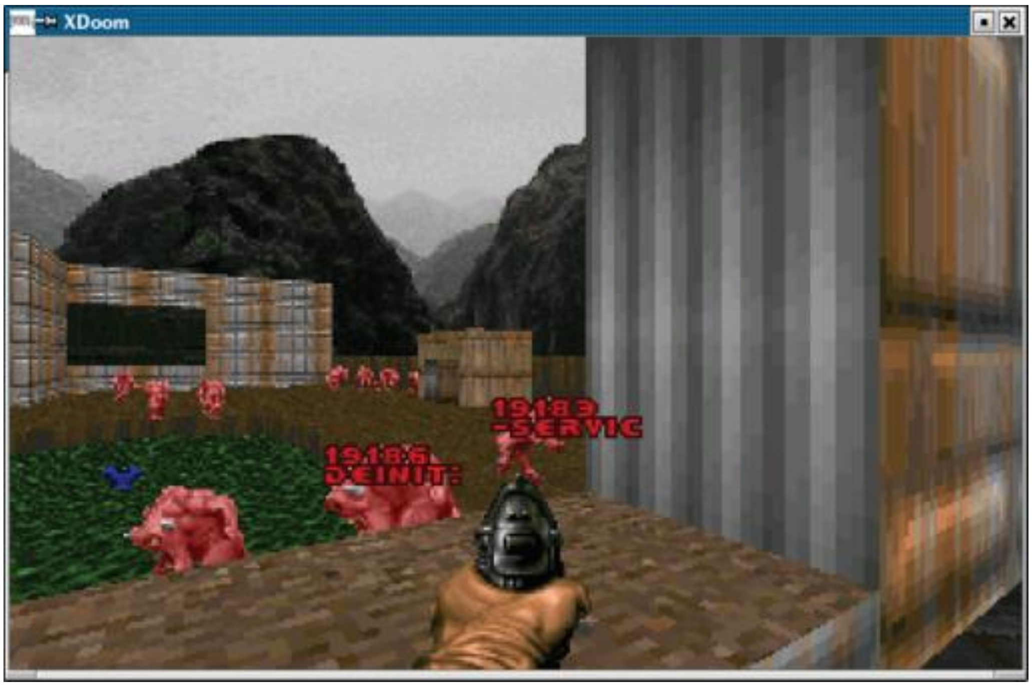


Figure 3. Killing Processes with a Vengeance: psDooM

Source tarballs are available from the psDooM web site, but the easiest way to install psDooM is to pick up the precompiled binaries. Installation is fairly simple: run the **install.sh** script:

```
tar -xzvf psdoom-2000.05.03-bin.tar.gz
cd psdoom-bin
su -c "./install.sh"
```

An IWAD is required to run psDooM, specifically Doom 1, Doom 2 or Ultimate Doom. The shareware Doom 1 IWAD also will work. If you don't happen to have your own Doom WAD, you can download a copy from the www.doomworld.com site. I visited the site and picked up a copy of the file:

```
unzip shareware_doom_iwad.zip
su -c cp DOOM1.WAD /usr/local/games/psdoom/doom1.wad
```

That's it. Now, we are ready to run psDooM:

```
cd /usr/local/games/psdoom
./psdoom -2
```

Notice the -2 option above. By default, you'll find the screen quite small. This option increases the size of your default screen. If you've never played *Doom* before, I must warn you, *mes amis*, that it can get a little violent. Monsters and bad boys roam the halls, and your life hangs in the balance. Wounding a process monster is equivalent to renicing that process (**renice +5**). Keep shooting, and you will kill the process. System permissions are honoured, however. You can kill a monster process that belongs to another user (or the system), but it will be resurrected. Only your own processes will *stay* dead.

Perhaps you might want to consider *not* doing this as root *ever*, and certainly not on the corporate server.

Once again, *mes amis*, it looks as though the clock has been racing toward closing time. It has been wonderful having you here at *Chez Marcel*. I certainly hope you enjoyed your exploration of process administration. I must admit I am still a little shaken from my psDooM experience. Perhaps a little more wine to soothe the nerves. François, if you would be so kind as to refill our guests' wineglasses and, of course, *mine*. Until next month. *A votre santé*! *Bon appétit*!

Resources

Marcel Gagné lives in Mississauga, Ontario. He is the author of Linux System Administration: A User's Guide (ISBN 0-201-71934-7), published by Addison-Wesley (and is currently at work on his next book). He can be reached via e-mail at mggagne@salmar.com.

Configuring and Using an FTP Proxy

Mick Bauer

Issue #104, December 2002

Mick shows you how to add a layer of security between the bad guys and your public FTP servers.

Running a public FTP site securely can be difficult. Taking full advantage of the security features supported by your FTP server application of choice can be a chore, and even then there's a good chance that sooner or later vulnerabilities will come to light making all that work for naught. So what else can you do?

One important technique is to run an FTP proxy on your firewall. Whereas the standard Netfilter code in the Linux kernel only inspects packets, an FTP proxy lets your firewall act as an intermediary in all FTP transactions. This increases your protection against buffer overflows and many other kinds of FTP attacks. It also allows you to restrict which FTP commands are executed by FTP clients.

This month I explain how to run SuSE's free (and non-SuSE-Linux-specific) Proxy-Suite FTP proxy on your Linux firewall, adding transparent but strong protection to all your FTP transactions.

Getting and Installing proxy-suite

If you run SuSE Linux, you can install the package proxy-suite, which installs a binary copy of ftp-proxy along with its configuration file and startup script. If you wish to use ftp-proxy as a transparent proxy, or if you want ftp-proxy to perform LDAP authentication, you'll need the latest version (1.9 as of this writing).

To run the latest version or use ftp-proxy on non-SuSE distributions, your best bet is to compile it yourself from source code, available at ftp.suse.com/pub/projects/proxy-suite/src.

Building from Source

Complete instructions on building and installing ftp-proxy are provided in the file INSTALL. By default, the configure script will check for libwrap, libldap and whether your system supports regular expressions. On my Red Hat 7.3 system, libwrap was present but caused a compile-time error, so I disabled libwrap like this:

```
# ./configure --without-libwrap
```

and ftp-proxy compiled properly. However, this wasn't necessary when I compiled ftp-proxy on my SuSE 7.1 system (obviously, SuSE's and Red Hat's libwrap packages differ).

After building ftp-proxy and installing it and its documentation, you'll probably want a startup script for your new proxy. Included with ftp-proxy's source (in the directory ftp-proxy/) is a sample script, rc.script, which is explained in the accompanying file rc.script.txt.

On SuSE systems, you simply can copy rc.script to /etc/init.d and optionally create a symbolic link to it from /usr/sbin. Rename the script /etc/init.d/ftp-proxy, and name the symbolic link /usr/sbin/rcftp-proxy. If you run SuSE 7.x, you'll also need to add this line to /etc/rc.config:

```
START_FTP_PROXY="yes"
```

For non-SuSE distributions, the example rc.script will need to be heavily tweaked, because much of it is SuSE-specific. Look at other scripts in your distribution's init.d directory for examples. Once you've figured out how, I strongly encourage you to send your hacked script to Marius Tomaschewski (mt@suse.de), one of the major contributors to FTP-Proxy, so others may benefit from your brilliance.

Configuring ftp-proxy

Once you've installed ftp-proxy from source or from a SuSE package, it's time to configure it. Most configurable parameters are kept in /etc/proxy-suite/ftp-proxy.conf (or, if you installed from source, in /usr/local/etc/proxy-suite/ftp-proxy.conf). Before diving into ftp-proxy.conf, however, you've got a couple of odds and ends to attend to.

First, you need a new, unprivileged user account for the proxy dæmon to use. On my system I created such a user, ftpproxy, like this:

```
bash-# useradd -u 65500 -g nogroup -d
/var/ftp-proxy/rundir -s /bin/false ftpproxy
```

No one should log in as this user, so be sure also to put an asterisk in the password field of the proxy user's line in /etc/shadow:

```
ftpproxy:*:12345:0:99999:7:0::
```

Next, you'll need to build a chroot jail in which ftp-proxy's child processes can work. For SuSE users this is easy; ftp-proxy's startup script will do this for you if invoked with the chroot command:

```
bash-# /etc/init.d/ftp-proxy chroot
```

Even if you don't run SuSE, it's fairly simple to reverse engineer the example script (the rc.script mentioned earlier) to figure out how to do this. The long and short of it is that the customary ftp-proxy chroot jail is /var/ftp-proxy/rundir, and it should contain copies of the libraries and files ftp-proxy uses, plus its own dev/log special file to which your local syslog dæmon can listen.

To point your syslog dæmon to the chrooted log device, simply add an -a parameter to its startup script so that syslog is started:

```
syslog -a /var/ftp-proxy/rundir/dev/log
```

On SuSE systems the customary way to do this is in /etc/rc.config via the SYSLOGD_PARAMS variable. You can specify multiple -a statements if, for example, you're also receiving logs from a chrooted named.

Firewall Primers

ftp-proxy.conf

And now, finally, it's time to configure your proxy dæmon. As I mentioned, this is done in the file ftp-proxy.conf, which resides either in /etc/proxy-suite or in /usr/local/etc/proxy-suite. You may be confused or annoyed by SuSE's use of the term "suite" to refer to a single application. Hopefully, additional proxies will be completed soon, and if they're as useful as ftp-proxy, I, for one, will forgive them for this minor conceit.

The quickest way to explain this file is to list a brief example and dissect it (see Listing 1).

Listing 1. ftp-proxy.conf

The first parameter, ServerType, determines whether to run ftp-proxy as a standalone dæmon or from inetd. Although I've been calling it a dæmon, ftp-proxy can be run either way. I personally avoid running inetd or even xinetd on my public servers, because that way I don't need to disable the unnecessary things that tend to get run by default, and because of the performance benefit of running things as dæmons. If your needs are different, you can set ServerType to inetd (which also works if you run xinetd rather than inetd).

User and Group, obviously enough, determine the UID and GID under which ftp-proxy runs after initialization. It's a good idea to set these to an unprivileged UID and GID in order to lessen the consequences of an attacker somehow hijacking an ftp-proxy process.

LogDestination specifies where ftp-proxy should send log messages. This can be either dæmon (the local syslog facility), a file or a pipe. LogLevel determines the quantity of information to be logged; for most users the default of INF is best, but DBG (the maximum setting) is useful for troubleshooting.

PidFile tells ftp-proxy where to store the process ID of its master process. This is used by the startup script when it's invoked with the stop command and upon system halt. It isn't used, however, if ftp-proxy is run in inetd mode.

ServerRoot specifies the path to ftp-proxy's chroot jail. Leave it commented out if you don't want to run ftp-proxy chrooted (see the “Problem with 1.9 and chroot” Sidebar).

Problems with 1.9 and chroot

Transparent Proxying

The next three commands in Listing 1 are important. They determine whether your proxy will be transparent. In most situations, a transparent proxy is preferable. End users won't need to configure their FTP client software to explicitly support the proxy. To achieve this, ftp-proxy works in conjunction with the kernel's Netfilter code, which redirects FTP packets to your proxy dæmon rather then sending them to the host to which they're actually addressed.

When ftp-proxy receives FTP client packets that have been redirected in this way, it uses their destination IP as the destination of the new FTP connection it initiates to the desired FTP server. The parameter DestinationAddress specifies the default destination to use.

If you want to allow users to use the proxy non-transparently, i.e., by initiating their FTP sessions directly to the proxy, set the parameter AllowMagicUser to “yes”, but I do *not* recommend doing so if your proxy is to be used by external users, as in the case of a public FTP. AllowMagicUser will cause your proxy to act as an open proxy that external users may use to connect to *other*, *external* FTP servers, possibly for the purpose of attacking them.

If you've configured Netfilter to accept connections to the proxy from trusted (internal) users only, however, and you set AllowMagicUser to “yes”, users will be able to specify their FTP destination by attaching it to their user name with an @ sign, e.g., mick@ftp.wiremonkeys.org. AllowMagicUser may be used regardless of whether AllowTransProxy is set to yes or no. But note that if it's set to no and AllowMagicUser is too, *all* FTP sessions will use DestinationAddress.

Other parameters include MaxClientsString and DestinationTransferMode. See the ftp-proxy.conf(8) man page for the complete list and for more information on the ones we've covered here.

Configuring Netfilter for Transparent Proxying

For transparent proxying to work you need to use iptables to redirect FTP packets to the local proxy (i.e., you need to run Netfilter on your proxy host, which this article assumes you're doing), and of course, you'll need rules allowing FTP connections to and from the proxy. You will *not*, however, need any rules in the FORWARD chain.

First, you'll need to load several modules for your Linux 2.4 firewall to support transparent proxying: ipt_conntrack_ftp and ip_nat_ftp are required for FTP connection tracking; ipt_REDIRECT is required for the REDIRECT rule target. Most distributions' stock 2.4 kernels include these modules.

Once the modules are loaded, you can add firewall rules like these to your Netfilter startup script (Listing 2).

Listing 2. iptables: Commands for Transparent FTP Proxying

The first two commands of Listing 2, instruct the firewall to redirect all packets received on its external and internal interfaces (eth2 and eth0, respectively) that have a destination port of TCP 21 (the FTP server port). Note that these packets won't be rewritten (mangled) in any way; they'll simply be redirected to the local FTP proxy dæmon.

The third and fourth commands in Listing 2 tell the firewall to accept all incoming packets sent to TCP port 21 of the public FTP server (where the variable PUBLIC_FTP contains its IP address) and all incoming FTP packets sent by internal users (where the variable INTERNAL_HOSTS contains an IP range in CIDR notation, e.g., 192.168.99.0/24). Per the first two lines, any packets matching lines three and four will be diverted to the local proxy.

The fifth and sixth lines in Listing 2 allow the local ftp-proxy dæmon to initiate proxied FTP connections to the specified public FTP server and to external FTP servers (i.e., hosts reachable from its external Ethernet interface, in this example, eth2).

The lines in Listing 2 do *not* form a self-contained Netfilter rulebase. They represent the lines you could add to an existing script already properly configured for NAT, etc., and already containing definitions for the variables PUBLIC_FTP and INTERNAL_HOSTS. It's good practice to use custom variables like this to make your rules more readable.

Restricting FTP Commands

Now we return to ftp-proxy.conf (Listing 1) and one of ftp-proxy's most important features: ValidCommands. This is a comma-delimited list of FTP commands the proxy will allow. The list may span multiple lines if you end each line (except for the last) with a backslash (\). In the ValidCommands statement at the bottom of Listing 1, ftp-proxy has been configured to allow FTP directory navigation commands (PWD, CWD, CDUP) and FTP read commands (LIST, NLST, RETR), plus some additional administrative commands such as MODE, PORT and PASV.

Space does not permit me to explain all of these in depth, other than to say that these aren't end-user FTP client commands; they're FTP protocol commands as specified in RFC 959 (see ftp.isi.edu/in-notes/rfc959.txt). These are the commands that FTP client and server applications use with each other. See Table 1 for a summary.

Table 1. FTP Commands Specified by RFC 959

One limitation of ftp-proxy is that it isn't possible to set different command restrictions for external users than for internal users. Be careful, therefore, with ValidCommands. If your internal users need to send files to FTP servers, you won't be able to restrict the STOR or STOU commands (i.e., you'll need to include them in ValidCommands), which means you'll need to make sure your read-only public FTP server is itself configured to disregard them.

That isn't such a bad thing. Regardless of how ftp-proxy is configured, you still need to configure your FTP servers to protect *themselves* as much as possible.

Conclusion

An FTP proxy adds an important layer of security between the bad guys and your public FTP servers. I've shown you the basics of setting up a transparent FTP proxy using SuSE's proxy-suite, but it supports many other worthwhile features we haven't covered here. See the Resources section for pointers to additional information. Good luck!

Resources

Mick Bauer (mick@visi.com) is a network security consultant for Upstream Solutions, Inc., based in Minneapolis, Minnesota. He is the author of the upcoming O'Reilly book Building Secure Servers with Linux, composer of the “Network Engineering Polka” and a proud parent (of children).

On System Administrators

David A. Bandel

Issue #104, December 2002

Is a basic understanding of the way things work lacking in this generation's system administrators?

System administration means different things to different people. Some will tell you anyone who performs any kind of administration on a system is a system administrator. But I have a hard time calling someone who can only add users via a GUI, a system administrator. To me, a system administrator is someone who understands what goes on behind the scenes. I recently interviewed several folks for a position in my company who claimed to be network administrators. Most had no idea what a netmask was or why it was needed. Or, they knew what ARP was, but had no idea how ARP and IP interrelated. One even knew the OSI model, but where ARP, IP and TCP and UDP came in he didn't know. Understanding how things work makes troubleshooting easier and defines a true system or network administrator. Here are some tools that can help and a couple of fun programs too.

reportdhcp.pl www.omar.org/opensource/reportdhcp

Take a dash of Perl, a little knowledge of where your configuration and logging files are, a little slicing, dicing and formatting, and you have a *very* useful tool for finding out about your DHCP leases via your web browser. The instructions are readable and simple. In two minutes time you can search IPs, MAC addresses, client names, get stats on the server and more. Requires: Perl, a web server and browser.

pgmaint sourceforge.net/projects/pgmaint

If anyone out there is using Postgres for logging Snort data (especially on a heavily trafficked site), you probably already know the importance of cleaning out your database from time to time, and vacuuming and analyzing it. If you have several of these databases, the cleanup chores can get a bit tedious.

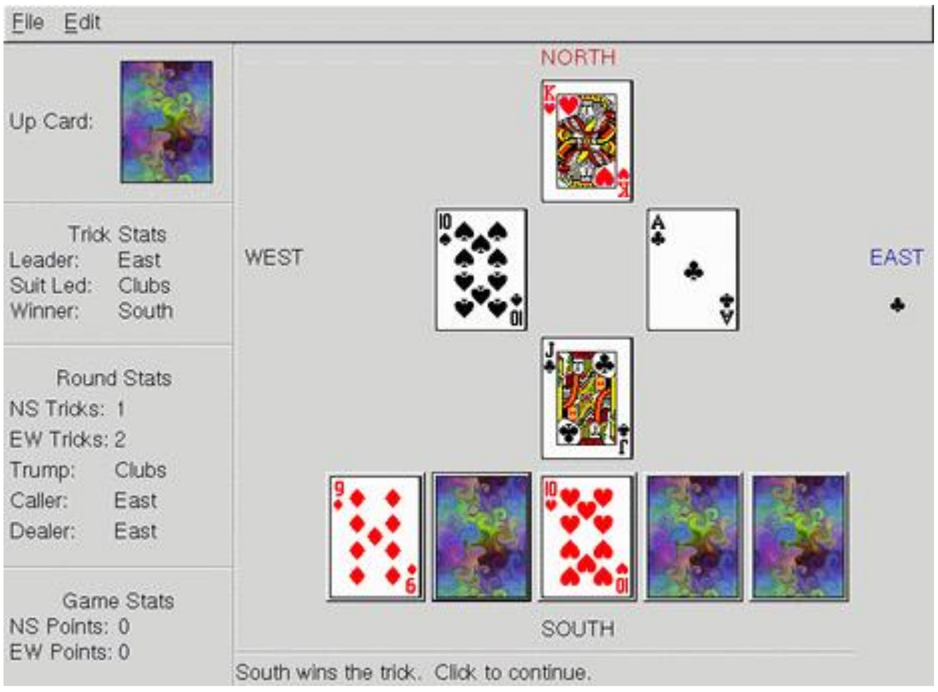
pgmaint can handle all this for you, even via cron. Requires: Perl, Perl modules DBI, Config::Simple and Getopt::Mixed.

Katoob www.arabeyes.org/project.php?proj=Katoob

This editor handles several formats well. Designed for either English or Arabic, it has menubar icons to change from left- to right-justified text and more. While a bit heavy on library and memory usage, anyone already running X won't particularly notice. Requires: libgtk-x11-2.0, libgdk-x11-2.0, libatk, libgdk_pixbuf-2.0, libm, libpangoxft, libpangox, libpango, libgobject-2.0, libgmodule-2.0, libdl, libglib-2.0, glibc, libX11, libXi, libXft, libXrender, libXext and libfreetype.

Euchre sourceforge.net/projects/euchre

If you like *Euchre*, this is a nice version of the game. The AI players have three configurable levels of play, and the author includes instructions for those unfamiliar with the game. Play is fast and easy. Requires: libgtk, libgdk, libgmodule, libglib, libdl, libXi, libXext, libX11, libstdc++, libm and glibc.



Trick Stats
Leader: East
Suit Led: Clubs
Winner: South

Round Stats
NS Tricks: 1
EW Tricks: 2
Trump: Clubs
Caller: East
Dealer: East

Game Stats
NS Points: 0
EW Points: 0

South wins the trick. Click to continue.

A jack of the trump suit beats an ace? Not a bug, a *Euchre* rule.

Childsplay childsplay.sourceforge.net

I'd review more children's games if I could find them because my children are always looking for computer games. My wife thinks our seven-year-old daughter doesn't need to be playing *Quake*, but just how many hours of Barbie.com games can a child play until boredom sets in? Not quite up to *gcompris*, but unencumbered of the megs of GNOME libs *gcompris* requires, *Childsplay* is shaping up to be a good game for your little ones. Requires: Python and pygames.

DNS Sleuth atrey.karlin.mff.cuni.cz/~mj/sleuth

This Perl script can be run either from a command line or via a web server with an included CGI script. It checks the domain name provided for compliance with the RFCs and reports errors with a reference to the appropriate RFC paragraph, so you can read what's broken and why, and hopefully fix it. Requires: Perl, Perl module Net::DNS and optionally a web server CGI. Until next month.

David A. Bandel (david@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

The Ethical System Administrator

Lawrence Rosen

Issue #104, December 2002

On the question of making illegal copies of proprietary software—why should we care?

My editor asked me an embarrassingly difficult question a while back and I've been struggling for an answer. It forced me to consider questions of ethics as well as law. He asked, “What should a system administrator do when he discovers that others in his company are making illegal copies of licensed software?”

Under the laws of most states, nobody has a duty to be a good samaritan. You may remember the Biblical parable of the passerby who came to the aid of a Jew who had been robbed, beaten and left to die on the roadside. The kindness of the Samaritan was particularly admirable because Jews and Samaritans generally were enemies. Under the law, a good samaritan is someone who voluntarily renders aid to another in distress when under no duty to do so.

Since a system administrator doesn't have a legal duty to be a good samaritan, she owes no legal duty to the software vendor who supplied the software to do anything to stop the illegal copying.

Many companies have internal codes of conduct that obligate employees to report illegal conduct to their managers or company executives. If a system administrator fails to report illegal copying of software, she may be in violation of company policy and subject herself to possible termination. I always advise an employee to conduct herself according to the company's own published rules.

If a company doesn't have a published code of conduct, the system administrator should try to understand her company's implied code. Is open cheating tolerated within the company? Do senior managers condone or encourage illegal conduct? I once sued a company on behalf of an employee who was fired for refusing to dump toxic waste into the public sewer system leading to the San Francisco Bay. Unfortunately, as I came to discover, such behavior was sanctioned and encouraged by the most senior executives of the company. In such a company, reporting illegal behavior would be useless.

Would it be useless to report that your company makes unauthorized copies of software? If you were a system administrator for such a company, would you want to continue working for them? Would you want to stay at a company that openly encouraged copyright infringement?

What about a system administrator who herself was making illegal copies of software? She cannot excuse her own illegal act by claiming that her employer told her to do so, because an employee cannot knowingly violate a law and blame her employer for it.

As a practical matter, of course, if there were a lawsuit or a criminal complaint about illegal copying, it probably would be the most senior company official responsible who would be punished, not the low-level employee who was merely following orders. But I would still advise the system administrator not to risk liability by committing an illegal act. It is safer simply to refuse to do so or to quit.

An employee who is asked to perform an illegal act and refuses to do so is protected from retaliation in many states. An employee who reports illegal acts to more senior management or to an appropriate government agency is often protected by “whistle-blower” laws. A company can be ordered to pay substantial damages, including back pay, for retaliating against whistle-blowers.

But what about the ethical dimension to the question posed by my editor? Why do we care if a company makes unauthorized copies of a proprietary vendor's computer software? We don't have a good samaritan duty to help them, so why should we bother ourselves with their problem? After all, one major objective for the Free and Open Source movement is to make software available that can be freely copied, modified and distributed. By definition (e.g., the Open Source Definition, www.opensource.org/docs/definition.php) it is never illegal for a company to make as many copies as it wants of free and open-source software.

If proprietary software companies don't share our ideals, that's their problem and not ours? Right? I'm not so sure that is ethical.

One reason the open-source software model works is that we can rely on the copyright law to enforce our licenses. When a company takes GPL-licensed code and converts it into proprietary software, for example, we can assert the copyright law and sue them for copyright infringement to prevent them from misusing the software. How, then, can we ignore that same law and make illegal copies of proprietary software? Is it ethical to rely on a law to protect our own interests and to violate that same law when it comes to someone else's?

This is one reason why I encourage everyone NOT to make illegal copies of proprietary software. I am willing to let the copyright laws work for me, even as large proprietary software companies use that same law to compete against me.

As an ethical attorney—or at least an attorney who strives to be ethical—I must advise everyone to obey the law. Don't make unauthorized copies of software, and don't just sit idly by when you see others in your company doing so.

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 the law of 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.

email: lrosen@rosenlaw.com

Lawrence Rosen is an attorney in private practice, with offices in Los Altos and Ukiah, California (www.rosenlaw.com). He is also corporate secretary and general counsel for the Open Source Initiative, which manages and promotes the Open Source Definition (www.opensource.org).

Archive Index Issue Table of Contents

Advanced search

Identity as Business Opportunity?

Doc Searls

Issue #104, December 2002

The carrier and content industries aren't buying the Net's end-to-end argument. Doc suggests using an open customer ID standard to win them over.

In 1997, David Isenberg wrote in “Rise of the Stupid Network”, a widely circulated internal document at AT&T (www.rageboy.com/stupidnet.html), “The Internet breaks the telephone company model by passing control to the end user. It does this by taking the underlying network details out of the picture.”

The document cost David his job at AT&T. But he was right. The Net's end-to-endedness supports countless new efficiencies that go without notice because they happen in the background. GE Global eXchange Services (gxs.com), for example, processes over a billion transactions worth over a trillion dollars a year—all over the Net.

GXS President and CEO Harvey Seegers told me that the Net evened the balance of power between supply and demand. In the past, GE was in command of its relationships with its customers, he said; but that ended with the Net. Now even giant companies like GE are finding they can't muscle their customers anymore, and they're liking it, because now they have much healthier relationships with them.

After *Cluetrain* (cluetrain.com) came out, we heard the same kind of thing from a lot of other big companies, including Wal-Mart, Johnson & Johnson, Prudential and Ericsson. But a conspicuous silence came from the companies that actually carry the Internet. Verizon, AT&T, Qwest, Time Warner Cable, Comcast, Cox and SBC all try to talk Internet, but they still seem to be dreaming about subscription valves on content spigots. This dream has been a nightmare ever since carriers began to build broadband services around the same power asymmetries established by consumer marketing. To them, the Net was all big-to-small and few-to-many, like television.

The big “content” companies from the entertainment industry shared those fantasies. Unfortunately, they did get the Net's founding clues—and hated them. The last thing they wanted to see was billions of dumb consumers turned into smart customers—or worse, smart suppliers.

But in the long run (and it might be very long), the Net will beat big-to-small. When that happens, the question for the carrier and content companies will be no different from what it was in 1995: How can I make money here?

Arguing about end-to-end isn't going to do the job with Congress, and it probably won't do the job with business, either. What we need is implementation. I suggest identity (ID). All kinds of business opportunities open up once identity protocols are ready to support business innovations based on them.

To gauge the importance of ID, consider the matter of your own identities—those representations of yourself in the business world. How many of them are granted to you by outside organizations? How many are mostly beyond your control? Is it close to 100% in both cases?

What new business possibilities would open if your suite of identity representations were under your own control? What if your identity operated in the networked world according to protocols that granted your end as much power as the ones that make and distribute products and the ones that process transactions?

What new businesses would be possible if your identity suite had its own open APIs to let you choose what to do with your personal information—your location, your destinations, your interests, your preferences for anything and everything? What types of existing businesses suddenly will find all kinds of productive new ways to relate to customers? It's easy to imagine countless new businesses—and a lot less unwanted advertising, PR, promotion, junk mail and other annoying guesswork about what you, the consumer, might want.

Making this happen is what PingID.com has been up to in the year since it was founded by Andre Durand, who also cofounded Jabber.com. Like Jabber, PingID has a .org counterpart (PingID.org) that handles open-source development. (Disclaimer: I am on the advisory boards of both Jabber.com and PingID.com.)

PingID.org is working on a peer-to-peer browser or thin (embedded) client-based public digital identity infrastructure. It makes heavy use of XML, SOAP, XML-RPC and other standards that Jabber also trafficks in. At the helm of PingID.org's efforts is Bryan Field-Elliot. I recently asked Bryan how it's going. Here's what he told me:

> Looking back over the history of protocols and their implementations, I see two different routes we can take. One is the Gnutella route, where we trailblaze a new protocol that perfectly suits our vision of identity rollout. The other is the Apache route, where we do practical implementations of protocols that have already been developed. We're trying to balance both routes and see what happens.
>
> Say that PingID.org trailblazes a new protocol putting ultimate privacy control in the hands of the user—priorities, which are secondary at best for Liberty and Passport. Can we realistically expect any traction in the real world given the momentum behind the other identity protocols? Or instead, should we follow up on protocols developed by, say, the Liberty Alliance? Right now those protocols are not especially friendly to the individual, but someday they might be. There's a better chance of that happening if we're in the conversation. This is the fine line we're walking: a balance between idealism and pragmatism.

The choice Bryan presents is about implementation. But motivation to implement only comes when people start imagining all the cool ways this stuff can be put to use. Here are eight that have come up for Andre Durand, solely through his conversations with businesses that have taken an interest in building out ID infrastructure: 1) identity service providers (basically identity hosting services); 2) identity verification/authentication services (certificate/signatory services); 3) identity guardian services (companies that protect/monitor the use of your identity information); 4) identity interoperability services (companies that help transport an identity in one system to an identity in another system); 5) identity reputation companies (companies that help you maintain and monitor your digital reputation to ensure it is clean and accurate); 6) identity networks (this is what PingID Network is building); 7) a branded, quality assured network that facilitates business-to-business and business-to-consumer identity interaction; and 8) identity registrars (companies that “issue identities”).

There are also existing business categories that can make extensive use of a public ID infrastructure: banks, cell-phone companies, credit-card companies, retailers and our original two: carriers and content providers.

With an end-empowering ID infrastructure in place, many new carrier services to customers suddenly become imaginable, because each customer is much more than a name, a number and an address. Their identities are not only far more rich and detailed, but their behavior is far more active and involves far more choice. Rather than threatening carriers, it gives them an API to which they can address far more granular and flexible services, including bandwidth as symmetrical or asymmetrical as the customer requires.

When preference-rich and fully empowered customers show up in the content marketplace, eager to develop relationships with the artists who make movies and recordings, all the DRM issues that hog conversational airtime today get exposed as desperate attempts by distributors to control “consumer” behavior.

A sufficiently end-empowering digital identity infrastructure will make consumers evolve into customers. When that happens, all arguments about controlling consumer behavior become moot, and the end-to-end argument finally wins, big time.

Doc Searls is senior editor of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

VariCAD 8.2-02

Michael Baxter

Issue #104, December 2002

VariCAD is a serious, high-end 2-D and 3-D CAD tool for mechanical engineering design and competes favorably with tools costing substantially more.

VariCAD is well known as one of Central Europe's strongest players for CAD tools for mechanical engineering. Their software has been continuously improved under industrial use for 14 years and has users in 40 countries around the world. They are a friendly company, willing to interact with you to solve a problem or glean feedback about their tool.

The product reviewed here was VariCAD 8.2-02, which was tested on a Red Hat Linux 7.3 system. VariCAD supports binaries for Red Hat, Mandrake and SuSE Linux distributions, as well as for Microsoft Windows.

Despite the low price (less than $400), VariCAD is a serious, high-end 2-D and 3-D CAD tool for mechanical engineering design and competes favorably with tools costing substantially more. In fact, if you have a design team, discount pricing is also available.

The tool capabilities include 3-D modeling and 2-D drafting, support for metadata, such as title blocks, and linking libraries of machine parts to designs. VariCAD is a fast GUI tool based on OpenGL. High-performance graphics are vital to a mechanical CAD tool, where a designer's time cannot be wasted on slow screen redraws for changes in views. The 3-D interface is responsive on my test system, a KDS Valiant 6480CiPTD laptop with an 800MHz Intel PIII, 512MB of RAM and a Trident CyberBLADE Ai1 video chipset.

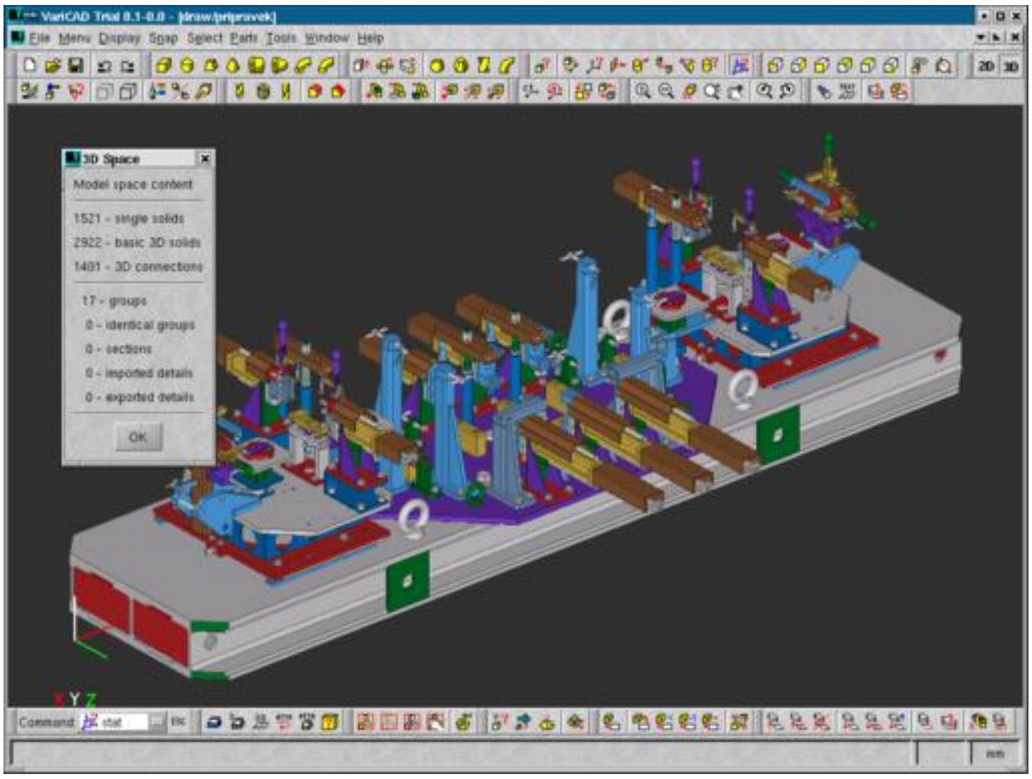
However, due to the numerous GUI icons, there is a learning curve to get up to speed on the iconic language within the tool. VariCAD supports a nice on-line tutorial to help you familiarize yourself with the menu flow for icons and available features. Going through this really aids in understanding how the iconic language works. The language makes a lot of sense, and screen real estate for icons is well suited for providing control over a strongly visual environment. This all seems appropriate for a tool in the 2-D/3-D field.

However, VariCAD does *not* have a user-accessible scripting language. This is an important usability feature for users who need to automate key design tasks, such as making variations automatically given a starting or genus mechanical design. The tool will not serve well for users who need or already have large scripting applications for automating their design tasks.

The supported capabilities *without* scripting, however, are enormously powerful. In CAD tools, parts libraries are everything, and VariCAD is well stocked with libraries that can generate thousands of parts based on ANSI or DIN industry standards. This feature greatly simplifies generating components for designs. For instance, the components might be fasteners, building blocks in large designs that incorporate custom metal machined parts. It is convenient and fast to create screws, keys, springs, bearings and numerous other types of parts. All of the components used in a design may then be linked automatically to a very convenient Bill of Materials (BOM) generator.

The file-locking feature supported by the tool is a rather crude form of access control. If a user attempts to access a locked file, typically for an in-use document, contact information is displayed for the user desiring access. It certainly might make sense within a design organization for maintaining access rights for key drawings, but in this tool, it is only effectively operable in limited contexts—during edits. Adding GUI features to the tool to provide access using a minimal subset of a real version control system like GNU CVS would have been much more effective. This would have enabled significantly more capability for supporting access rights as well as multiple-person design teams.

The general feel of the tool is that of working on the level of a totally visual-oriented tool. The iconic hierarchies are specific and work quickly. One gets used to automatically created elements and working them into the design flow seamlessly.


VariCAD Trial 0.1-0.0 - [draw/pripravek]
File Menu Display Snap Select Parts Tools Window Help
2D 3D
Command:
X Y Z

```
3D Space
Model space content

1521 - single solids
2922 - basic 3D solids
1401 - 3D connections

17 - groups
0 - identical groups
0 - sections
0 - imported details
0 - exported details

OK
```

Sample Screenshot in 3-D Space

One of the best features of VariCAD is being able to work directly in 3-D to design an object. This mirrors experience in the real world—objects have mass, dimension and textures. From here, 2-D drawings can be created automatically. This is where the high-performance graphics shine in terms of usability. This performance is most likely due to the use of OpenGL as an infrastructure technology.

Also significant is the fact that VariCAD collaborates with the OpenDWG alliance. Getting CAD data into both standard and alternative digital forms is important for manufacturing, component testing and numerous other unique situations in mechanical engineering. The tool supports DWB, DWG, DXF and IGES data formats for maximum interoperability of mechanical datasets to/ from other tools or manufacturing machines and to support Internet-based manufacturing. VariCAD also conveniently supplies a multiplatform drawing viewer free of charge.

The on-line (English) documentation supporting the project ranges from fair to good. And though it does contain grammar and spelling errors, the documentation generally allows you to grasp what you need to know. It's based on HTML, so it runs on a browser. Some professional users in North America might have preferred that this information be contained in a PDF instead—it's common to print a small number of relevant manual pages on paper, for working alongside a large screen monitor. But with Linux, it is also easy to have multiple desktops or a two-headed display. The company has taken a somewhat conservative approach to platform issues with their tool, supporting the KDE desktop only.

VariCAD is an impressive value proposition for a powerful Linux desktop application that supports 3-D and 2-D mechanical CAD engineering.

Product Info./Good/Bad

Resources

email: mab@cruzio.com

Michael Baxter has been working in computer technology since he was nine, imprinted by a 1969 viewing of 2001: A Space Odyssey. He is an experienced computer architect, system, board and FPGA logic designer. Michael holds ten US patents in computer architecture and logic, plus five patents as a co-inventor. His interests also include hiking, amateur radio and programming in Python.

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #104, December 2002

Readers sound off.

Now, That's Support

I had a dual-processor system built by Los Alamos Computers in February 2002. The system would lock up shortly after being powered on. I pulled the cover off and saw that one of the CPU fans was not spinning. LAC promptly sent out two new CPU fans with instructions for installing them. I installed the new fans, but the system was no more stable. Gary at LAC suggested that maybe a processor had been damaged from overheating, and they asked me to return the whole system. I received the system back shortly thereafter and had been happy ever since—until around August 19, that is. The system started locking up again. Larry started to suspect the Tyan 2460 motherboard and recommended returning the system to him so he could replace the board with an ASUS A7M266D. They also put in a quieter case and CPU fans, with my approval, again at no charge. I now have the system back, with a new motherboard and all new case and CPU fans. It is so quiet now, I can run it in the dining room and my wife does not mind. LAC truly stood behind their product.

—Chris Hajer

Hooray for AOLserver and ACS

This series (beginning with “Introducing AOLserver”, September 2002) will make my *LJ* subscription worthwhile. Reuven, you can quote that.

—Lamar Owen

LJ T-Shirt's Asian Tour

Thought you might enjoy this picture taken on the Great Wall in China. Fitting, I thought, for the “In a World without Fences...” message on the *Linux Journal* T-shirt. I was in Asia for a series of Linux meetings across Beijing, Seoul, Hong Kong and Tokyo (I work for a large, three-letter acronym company that is heavily focused on Linux). During this trip, my *LJ* shirt was very popular at all locations—washed periodically, of course.

—Bill

The Great Wall

Red: the New Beige?

I just finished reading your article on the Ultimate Linux Box [*LJ*, September 2002] and am happy to say that I have the exact components you mentioned. I have not installed Linux as of yet, but as proof of the SC760 case being paintable and modifiable take a look at what I did to my case.

—Will Mendez

Quiet Linux Box Tips

I'm writing in response to Mr. Scarbrough's letter on page 6 of the October 2002 issue: “Turn That Thing Down!”. I too was terribly annoyed with the constant wind-tunnel sound coming from my always-on PC and found the following solution. My first step was to install LMsensors so I could read the temperature sensors on the motherboard, an ABit VP6. Without any changes, the CPU temps were about 95-98°. I then underclocked the CPUs down to 466MHz and reduced the CPU core voltage to only 1.30 volts. This reduced the CPU temp to almost room temperature.

Then, I disconnected both CPU fans and watched the temp climb up to 109-111°. I then rewired the three-wire CPU fans to run at 5 volts with the third wire disconnected. At this speed, the CPU fans were dead silent and CPU temp dropped back to room temp.

Next, I replaced my NVIDIA GeForce2 card and its noisy fan with an older Diamond Viper V550 video card without any fan at all. I'm a housewife, not a gamer, so I'm not missing anything here.

Next, I rewired the power supply fan to use only 5 volts instead of 12. This made the power supply fan dead quiet. To be on the safe side, I added another fan wired to the 5 volt supply on the bottom of the case to draw cool air into the case. Good 12 volt fans running at 5 volts make practically no sound.

Running with the above mods, my computer is near dead silent, and I can't hear anything unless I put my ear just a few inches from the case. The CPU temp is a nice cool mid-80s even when compiling or playing music. I run Apache with PHP, MySQL, cidd, SSH, e-mail, Galeon and dozens of other off-the-net and custom programs. I really don't notice any difference between running at 933MHz and 466MHz, and even in the dead of night, the only indication the computer is running is the little light on the front.

On a side note, I discovered with LMsensors, I can read the status of the “chassis intrusion switch”. I'm not sure how I'll use this yet, but you can be sure I'll come up with a more interesting use for this than detecting “chassis intrusion”.

—Mrs. Kim Genrich

Thanks for the tips. (Readers, remember that only people trained in high-voltage safety should open or modify power supplies.)

—Editor

Archive Index Issue Table of Contents

Advanced search

UpFront

Various

Issue #104, December 2002

LJ Index, Stop the Presses and more.

diff -u: What's New in Kernel Development

Multiple logical CPUs on a single physical CPU, or **hyperthreading** (HT), is a relatively new concept introduced with Intel's Pentium 4 Xeon that lately has been getting a huge amount of support in the Linux kernel. **Ingo Molnar** started it off in late August 2002, with a patch to add HT-awareness to the scheduler. Many patches have followed since then, and hyperthreading has become quite the mainstream feature.

khttpd is finally going away. After a long and bitter struggle from the moment of its inclusion, the kernel-based web server is gone from 2.5 and will not be back. In spite of the tremendous controversy surrounding this feature, it was not the flame wars that eventually sealed its doom, but the fact that user-space **Tux2** is much faster. There is even some talk of putting Tux2 into the kernel as a replacement for khttpd, though many folks object to this on the same grounds that they originally objected to khttpd. Some patent issues also are holding up such an idea, and it looks as though no one really wants to bother fighting it out.

Several new system calls found their way into the kernel in August and September 2002. Among them, **clone_startup()** on x86 boxes reduces the number of system calls required for thread creation to one. Glibc's fancy pthread code is one big user of this call. The only problem is that the name has not quite been nailed down yet. YMMV.

One cute little tidbit: the **PC speaker** in post-2.5.31 kernels may now be used as a microphone. This is new and weird. As **Jos Hulzink** put it on the linux-kernel mailing list, “2.5.32 will go into the history books as the kernel that implemented voice recognition for all AT class computers....”

The struggle for a new kernel configuration system is ongoing. With CML2 apparently out of the running, several new configuration systems have emerged. Among them, **"kernel conf"** may be the most likely to succeed. **Roman Zippel** has been working on it steadily and claims his code is nearly completely usable. The rumor is kernel conf is likely to go into the main 2.5 tree, but no official announcement has come out yet.

A new tool, called **devlabel**, has surfaced from **Gary Lerhaupt**. It allows consistent access to storage devices via dynamic symlinks, with support for hot plugging. Simply plug in your device, and a symlink appears that may be used to access the device. Unplug the device and the symlink goes away.

—Zack Brown

LJ Index—December 2002

1. Number of wireless nodes discovered with an iPAQ by hackers flying in a four-seater Grumman over Perth, Australia: 95
2. Number of wireless nodes discovered by the same hackers using a Toshiba laptop: 92
3. Speed in MPH of the Grumman over the ground: 250
4. Altitude in feet of the Grumman throughout the flight: 1,500
5. Number of wireless nodes discovered by Phil Windley, CIO of Utah, flying in a Piper Turbo Arrow over Salt Lake City: 27
6. Number of encrypted nodes among those: 5
7. Speed in MPH of the Piper over the ground: 125
8. Altitude in feet of the Piper throughout the flight: 1,500
9. Linux percentage share of 110-million-unit desktop market: 2.7
10. Unit sales growth of Linux desktops in 2001: 47
11. Numbers of Linux desktops distributed for every one purchased: 12-15
12. Percentage growth of Linux new license revenue shipments over the last year: 28
13. Concurrent decline in new license revenue shipments for UNIX: 25
14. Number of Zumiez stores installing Linux desktops: 100
15. Estimated $-per-desktop savings over Microsoft alternatives at the Zumiez stores: 500

Sources

- 1-4: E3
- 5-8: Phil Windley

- 9-15: International Data Corp.

Stop the Presses: Generic PC, Your Identity

Scott McNealy dropped the first shoe at LinuxWorld Expo in August 2002. In his keynote address, Sun Microsystems' President and CEO said the company would be announcing a new Linux desktop at its SunNetwork conference in September. When the second shoe dropped at SunNetwork, it didn't appear to match the first. Rather than yet another Linux box, what the company announced was a desktop strategy meant to take advantage of the huge enterprise market for inexpensive and flexible no-name x86 PCs, and for cost savings in general.

The code name for the strategy is Mad Hatter. Here is how Curtis Sasaki, Sun's vice president of engineering, desktop solutions, explains it:

> What we're announcing is a complete package, not just a box. You get a combination of hardware, software, services and back-end middleware as well. The hardware is a desktop with a Linux kernel, GNOME GUI, integration of the Java 2 platform, Mozilla, StarOffice and the Evolution application suite from Ximian. The differentiating factor is integration. What you get with one of our boxes is enterprise-ready and scalable, with directory, calendar, messaging server and Java Cards for access control as well. [Java Cards are Java-enabled smart cards.]

Target customers are cost-conscious companies with large populations of “transaction workers”. But rather than selling droneware for cubicle hives, Mad Hatter's angle is all about individual identity.

“Today CIOs want to know exactly what it costs per user to have e-mail, calendar and a directory account—and what it costs for security”, Sasaki says. What Sun wants is for the enterprise to populate itself with Linux PCs that are personalized at authentication by the user's Java Card, and there appears to be a demand for this. Sasaki says:

> While customers are extremely interested in aggressively priced systems based on open standards, they are also interested in the most unique hardware aspect of this offering: the Java Card that allows the administrator to provision web sites and applications-based user authentication.

This brings us to a whole new classification and an acronym to go with it. “It's not a PC, it's an IdC—an identity computer”, Sasaki says. “Identity is a big deal. It's about getting access to your desktop no matter where you are, based on your credentials.”

Eric Norlin, an analyst at Digital ID World, provides some context:

> Corporate IT has become almost a pure cost center. Just about the only IT efforts actually saving money (while increasing privacy and security) are in identity management. You can expect identity to move forward while other services tread water, because identity has the real promise of converting IT from a cost center to a profit center. With Digital ID you have a real possibility for ROI on IT investments.

Sun's intent also is to make a single identity work outside and between companies as well as inside the user's company. Sasaki explains how:

> Liberty Alliance has an open spec developed by 115 companies from many industries. That spec answers the challenge of creating a way for users to sign on once for multiple services. You're going to see a lot of different Liberty-enabled services being able to utilize your identity securely. When there is a business relationship with an enterprise that also deploys Liberty-enabled identity—say, United Airlines or American Express—then you can actually move from one to another without re-creating your identity.

Because Mad Hatter is about a stack that runs on Linux, it can embrace a customer's existing x86 hardware, or it can respect the interests of countries like China, which insist on domestic manufacture. Hardware is solely one swappable component. Writes Shahin Khan, Sun's chief competitive officer, “It is set up so you can delete and replace any components that you do not want.”

Why now? According to Sasaki, the Linux desktop software stack is finally complete:

> In the last 12-18 months, LOTD (Linux on the Desktop) technology has really matured. A year ago it wasn't real. We couldn't deliver a complete desktop solution. Our office suite wasn't there. GNOME and KDE weren't mature enough. Mozilla wasn't at 1.0. Now GNOME 2.0 is pretty cool. StarOffice 6 is getting great traction. You've got a pretty nice product in Evolution. Now we're ready.

Sun will be putting together the first prototypes at its iForce centers before the end of the year and expects the first IdCs to start shipping in the first quarter of 2003.

—Doc Searls

They Said It

What we can buy today far exceeds what we need to keep up.

—Mike Prince, CIO at Burlington Coat Factory, in *Fortune*, explaining why he's asking for an IT budget cut after converting to Linux.

Innovation makes enemies of all those who prospered under the old regime, and only lukewarm support is forthcoming from those who would proser under the new. Their support is indifferent partly from fear and partly because they are generally incredulous, never really trusting new things unless they have tested them by experience.

—Nicolo Machiavelli

Microsoft has chosen to make the war against open source a religious one. In doing so it has just managed to highlight it further, meaning IT Directors who wouldn't have ever considered it are now thinking of moving over.

—Dan Kusnetsky, International Data Corp.

Typically people think about things such as BIND and Sendmail, which are very important; but there is a much more practical sense in which both free and open code helped spread the birth of the Internet. That's the decision made in architecting the browser that reveals source. The source is constantly available. People didn't learn HTML just by buying Tim (O'Reilly)'s books first. What they did was steal each other's web pages, made the tweaks they wanted and then bought Tim's books so they could figure out how to do it better the next time around.

—Lawrence Lessig

I say to you that the VCR is to the American film producer and the American public as the Boston strangler is to the woman home alone.

—Jack Valenti, before Congress, 1983

Basically, all the commercial people have their own agenda, and that's very healthy because you want to have these often-conflicting agendas to push the system into something that actually works for everybody.

—Linus Torvalds

If you know how the source code works, you are much more likely to be able to sort out your problems. You will be able to link the software with the OS better.

You won't have to spend so much on maintenance; the costs will be lower. It would also cost a lot less to develop the software in-house and get it to work the way you want.

—Retail CIO

Archive Index Issue Table of Contents

Advanced search

From the Editor

Don Marti

Issue #104, December 2002

An overview of the current issue.

Does Linux need a road map? In the bad old days, there used to be a marketing document called a “road map”. Information technology vendors would come down from on high and declare that during some quarter of next year, they would allow customers to start doing some task with their product, so customers should buy stuff now and it will work sometime in the future.

The theme of this issue is system administration, so here's a better road map for you. Go to maps.yahoo.com, and fill in 327 Ley Road, Fort Wayne, Indiana, USA. That's the address of Midwest Tool & Die, where our authors Craig Swanson and Matt Lung work. As you might expect, MTD is the very model of a modern manufacturer, with an ISO 9002 ce.pngication and a quality award from the State of Indiana, but they're also doing something that just might make it worth the 35 hours and 42 minutes Yahoo says it will take to get there from Silicon Valley. They're drawing their own road map.

Three years ago, MTD faced the need for a shared company directory and addressed it with OpenLDAP, Samba and other customer-friendly software (page 48). Instead of following a top-down road map from a single vendor, they made things work their way. Although one of MTD's secret weapons is a close working relationship with nearby engineering powerhouse Purdue University, companies are following similar strategies both in-house and in partnership with IT services firms large and small.

Here in Silicon Valley, it's starting to become apparent that the main reason for the so-called tech downturn is that all this stuff just started working. When your file server is fast and reliable, your client systems are capable, and the operating system is stable, you don't need to upgrade in search of a fix. Now that “tech” isn't a money pit you have to struggle with, you can really start to get some use out of it, and that's only a downturn if your business is selling flakiness. For those of us who actually get some use out of systems, it's a win.

LDAP is a central directory that doesn't put harsh demands on other server software and works with pretty much anything. Of course, the more stuff you put on your company LDAP server, the worse it is when it goes down. Jay Allen and Cliff White explain a simple way to keep LDAP going, no matter what happens to the master server (page 58).

If you're new to system administration and interested in who ran what, when—whether for security, improving service to the users or some other reason—check out Keith Gilbertson's article on page 66. Process accounting is an old-school UNIX feature, and it might just be on the certification exam.

No matter how much attention HTTP gets, FTP refuses to die, and Mick Bauer explains how to protect your inside FTP server from the outside on page 32. There's plenty of other good stuff in this issue, so wash the whiteboard, grab your business requirements and draw your own road map.

Don Marti is editor in chief of *Linux Journal* and number eight on pigdog's list of “things to say when you're losing a technical argument”.

From Activism to Drive Partitioning

Heather Mead

Issue #104, December 2002

No one could ever accuse our readers of backing down without a fight.

One of our goals as a web site news source is to provide our contributors with a forum for sharing and discussing business practices and corporate decisions that affect Linux users. Adam Kosmin, author of “The Toshiba Standoff” (www.linuxjournal.com/article/6318) took us up on our offer. He wrote about his frustration with the Toshiba Corporation and their refusal to grant him a refund for returning the copy of Microsoft Windows that came pre-installed on his laptop. He outlines well the runaround Toshiba's customer service gave him; he couldn't get a laptop without Windows, and they wouldn't give him a refund, even though the Microsoft End User License Agreement offers one. They told him if he wanted to pursue it, “they'd see him in court”. Excellent customer service, indeed.

More recently, Adam updated us about the creation of WindowsRefund.net (www.linuxjournal.com/article/6363), his attempt to reorganize the operating system refund movement.

Continuing with the activism theme, Doc Searls wrote about his visit to Beverly Hills to attend the Digital Hollywood conference (www.linuxjournal.com/article/6360). Perhaps the fact that he was almost the only attendee with a laptop should have been a clue about who the rest of the crowd was. Are we the only ones thinking that the industry most desperate to control the Internet doesn't even *use* the Internet? Like *Fast Food Nation*, Doc's article confirms that, yes, it *is* that scary behind corporate doors. But he reaffirms the fact that the next generation of Hollywood blockbusters are all being made on Linux farms. So how long are the big guys going to fight the future when their own tech teams have switched sides?

Moving on to some of the technical articles, Leon Goldstein's review of Libranet 2.7 is titled, “Debian on Steroids” (www.linuxjournal.com/article/6358). A distribution based on Debian, Libranet's main appeal is it offers all the security and upgrade convenience of Debian, but with an easier installer.

Pat Shuff asks the important question, “How Many Disks Are Too Many for a Linux System?” (www.linuxjournal.com/article/6238). In light of the trend to recentralize resources and facilities, many system administrators are wondering exactly how big of a server they need to efficiently handle network traffic. According to Shuff, the answer to his title question depends on bandwidth, latency and addressability much more than it does on the OS running the show.

George Toft takes a look at “Using Logical Volume Management” (www.linuxjournal.com/article/5957), tries the LVM support in SuSE and Mandrake and reports on attempts to resize a ReiserFS partition on the fly. Now you don't have to re-install if you made your /var too small to log all the hits your web site is getting.

If you want to share your story of grassroots activism, or explain how you managed to connect your garage-door opener to the microwave to your office computer, send your article idea to Heather Mead at info@linuxjournal.com. And, be sure to check the *Linux Journal* web site often; new articles are posted daily.

Heather Mead is senior editor of *Linux Journal*.

Best of Tech

Various

Issue #104, December 2002

Our experts answer your technical questions.

Follow-up on How to Connect with Cox

These instructions for setting up Cox cable, which reader Matt Reynolds asked about in the September 2002 issue, are good for Tucson, Arizona, and most likely, for all Cox accounts. Cox uses DHCP and issues everybody what they call a CX number, which is needed to obtain an IP address from their servers. To make the DHCP client obtain an IP address, simply include that CX number in the dhcpcd command with the -I (client identifier) option:

```
/sbin/dhcpcd -I <cx number> <interface, (default
eth0)>
```

It would be a shame if Matt canceled his Cox account because he couldn't get Linux to obtain an IP address, as Cox in Arizona is excellent.

—Patrick Kellaher, kalmite@cox.net

Upgrading with RPM

I am trying to install KDE 3.0.3 on a fresh system, where neither KDE nor GNOME was installed, using RPMs. When I try to install qt-3.0.5-16.i386.rpm, I get:

```
[root@yeller rpms]# rpm -Uvh qt-3.0.5-16.i386.rpm
error: failed dependencies:
        libcups.so.2   is needed by qt-3.0.5-16
        libpng12.so.0  is needed by qt-3.0.5-16
```

So then I tried:

```
[root@yeller rpms]# rpm -qa | grep libpng
libpng-1.0.14-0.7x.3
libpng-devel-1.0.14-0.7x.3
```

```
[root@yeller rpms]# rpm -Uvh libpng-1.2.2-6.i386.rpm
+libpng-devel-1.2.2-6.i386.rpm
```

And I got another “failed dependencies” error. I've tried to resolve dependencies by working up from the lowest denominator, but I wind up in a spider web of RPMs that have more dependencies. So how do I install or upgrade without overwriting or losing something needed by something else?

—James Weisensee, itjayw@yahoo.com

Red Hat has created the up2date utility as an attempt to solve this problem. **up2date** requires that you register with Red Hat Network using the **rhn_register** command. **up2date** updates packages already installed on your system for which new versions have been released, most often relating to security patches. The **rpm -qa** commands you've tried are only querying your RPM database for packages already installed. If you want to find out which packages among a set of RPMs in a directory will provide a given file (such as libcups.so.2 or libpng12.so.0), then use a command more like:

```
for f in  libcups.so.2 libpng12.so.0; do
  for i in *.rpm; do
      rpm -qpl | grep -q $f && echo $i;
  done
done
```

to search each RPM package file listing for each of these filenames and to print the name of every package that contains said file(s). That won't always work (in some cases the desired file may be created by a package's postinstall script, for example). Also, some dependencies might not have filenames but are abstract identifiers that might be provided by any number of alternative packages.

—Jim Dennis, jimd@starshine.org

You often can get out of this kind of RPM upgrade mess by upgrading all the relevant RPMs at once, on the same command line. Just cursor up and keep adding the packages that RPM complains about to the **rpm -Uvh** command until it's happy. This works for removing codependent packages too.

—Don Marti, info@linuxjournal.com

Hello World? Hello Anyone?

I recently installed Linux, but I receive the following error when I try to run executables on my system:

```
bash: a.out command not found
```

How do I fix this problem?

—Manuel Sevilla, slickspick@yahoo.com

Looks like the directory you're in is not in your PATH. To run your newly compiled C program from your current directory, use **./a.out**.

—Robert Connoy, rconnoy@penguincomputing.com

Can I Print to This Thing?

I have a router with a built-in printer server. How can I print using the printer connected to the printer port on the router?

—Carl Maklad, cmaklad@tdxinc.com

When an appliance has a built-in printer server, that generally means it offers support for a specific list of network printing protocols (such as the MS Windows SMB printing services and/or the traditional UNIX lpd services). Assuming that your router supports lpd (one of the oldest and simplest remote printing protocols), you should be able to configure your Linux system to use that device as a UNIX remote (lpd) print spool. If you're using Red Hat, try the printconf or printtool utilities.

—Jim Dennis, jimd@starshine.org

Bug or Security Feature?

I have Red Hat 7.1 running IMAP, POP3 and POP2 from the University of Washington. The services do not accept client access with the login root and password, but other users are okay.

—Pedro Guedes, pmg01@netc.pt

The easiest thing to do is set up an alias for root in /etc/aliases to a non-root user. Don't forget to run **newaliases** after updating /etc/aliases.

—Christopher Wingert, cwingert@cwingert.qualcomm.com

You definitely should not try to login to an IMAP or POP server as root, especially if you aren't using SSL. You would be sending your root password in clear text for anyone to steal.

—Marc Merlin, marc_bts@google.com

How to Mount a Floppy

I have a ThinkPad 600 with an external floppy. How do I mount the floppy drive? (I was able to mount the CD-ROM no problem.) I've tried **mount /dev/floppy**, **mount /dev/fd0** and **mount /dev/fb0**, but none of those work.

—Zachary Grant Michael, zachary.michael@earthlink.net

See if your floppy drive is detected at boot time with:

```
dmesg | grep -i floppy
```

If you see a line

```
Floppy drive(s): fd0 is 1.44M
```

then your floppy is device /dev/fd0. To mount it, type

```
mount /dev/fd0 /mnt/floppy
```

—Usman S. Ansari, uansari@yahoo.com

I have a ThinkPad 570 with an external floppy, and it is detected as /dev/fd0. After you mount with the above command, you might want to customize /etc/fstab with the name of your floppy device and your chosen mount point in order to simply type **mount /mnt/floppy**. See man 5 fstab for how to do this.

—Don Marti, info@linuxjournal.com

Database with GUI for Teaching?

I am an Australian schoolteacher, and I am looking for a database to use with our Linux network. I want a database that is similar to Microsoft Access, because it is easy to use and teach to students.

—Ken Jordan, kwjordan@cedars.nsw.edu.au

There is a tool called pgaccess (<@url>pgaccess.org) that works with PostgreSQL and very closely resembles the functionality of Microsoft Access. Another alternative is to use the OpenOffice.org database connection facilities through ODBC.

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

Advanced search

New Products

Heather Mead

Issue #104, December 2002

miniHiPerCam, Black Lab v2.1, white dwarf linux v1.2 and more.

New Products

miniHiPerCam

American ELTEC unveiled an embedded Linux camera no larger than an ordinary surveillance camera. Available in both color and monochrome versions, the miniHiPerCam is based on an embedded version of ELINOS. The CMOS image sensor operates at a 640 × 480 pixel resolution and a frame rate of 15 or 30Hz. The built-in controller board uses a PowerPC 823 processor running at 50MHz. 16MB of memory is available for images and runtime data, and the OS with an embedded HTTP server and the application software are stored in 8MB of Flash memory. The camera has two RS-232 ports and one 10Mb Ethernet port.

Contact American ELTEC, 2810 West Charleston Avenue, Suite 57, Las Vegas, Nevada 89102, 702-878-4085, americaneltec.com.

Black Lab v2.1

Black Lab v2.1 is a cluster build and management suite for HPC clusters running Yellow Dog Linux. It offers single-click installation and configuration, automated updates through apt-get, a graphical user interface, control of multiple clusters and command-line control of all services. The new version incorporates BProc 3.0, which automatically migrates applications and shared libraries from the server to selected nodes. This setup allows each node to operate with a minimal installation on its local drive. Administrators can customize nodes with only the necessary software installed by designing their own server-side node images.

Contact Terra Soft Solutions, Inc., 117 West Second Street, Loveland, Colorado 80537, 970-278-9243, presales@terrasoftsolutions.com, www.terrasoftsolutions.com.

white dwarf linux v1.2

EMJ Embedded Systems announced the latest release of white dwarf linux, an embedded OS designed for 10MB of Flash memory and 16MB of DRAM. white dwarf linux supports any motherboard or single-board computer with IDE support and at least 8MB of RAM. It also supports CD and network installs. Version 1.2 includes kernel version 2.4.19, a graphical package-based installer, and GCC development tools for glibc 2.2.5. white dwarf linux works with the DIMM-PC 486 and 520, MOPS686+, CoolMONSTER and Tri-M MZ104 boards.

Contact EMJ America, 220 Chatham Business Drive, Pittsboro, North Carolina 27312, 800-548-2319 (toll-free), emjembedded.com.

EnCore M3

Ampro Computers has a new MIPS-based module for embedded systems. The EnCore M3 combines AMD's 400MHz MIPS32 Alchemy Au1500 chip with standard EnCore features to provide a complete standards-based CPU subsystem on a small form-factor module. The EnCore M3 is rated at 480 Dhrystone MIPS with a typical power consumption of less than 2.5 watts. The module features a 32-bit, 66MHz PCI Bus Interface. EnCore M3 is 100 × 145mm in size, and it includes two 10/100 Base-T Ethernet controllers and an AC97 audio interface. It supports up to 256MB SODIMM SDRAM and provides 2MB of Flash, two serial ports, two USB ports, a floppy disk controller, PS/2 keyboard and mouse ports, IrDA port and an ECP/EPP bidirectional parallel port.

Contact Ampro Computers, Inc., 5215 Hellyer Avenue #110, San Jose, California 95138, 800-966-5200 (toll-free), www.ampro.com.

Lindows 2.0

Lindows 2.0 is the latest OS distribution release from Lindows.com. Updates and new features for version 2.0 include a new GUI design, easily configurable support for more than 800 printers and the ability to use SMB print servers. Lindows 2.0 uses Netscape 7.0 as its e-mail client and web browser, featuring tabbed browsing and a pop-up blocking feature. For laptop users, Lindows 2.0 offers laptop power management and battery controls. For networking, the new release offers improved WiFi support and the ability to use Windows file servers.

Contact Lindows.com, Inc., 9333 Genesee Avenue, 3rd Floor, San Diego, California 92121, 858-587-6700, www.lindows.com.

NAG C Library Mark 7

Numerical Algorithms Group (NAG), an international association of computer scientists involved with math, statistics and 3-D visualization software, released a new version of its C library of over 850 mathematical functions. The NAG C Library includes functions for modeling and simulation, time series analysis and statistical routines for a broad range of software. The functions can be accessed and used from Linux and other platforms, and from a variety of languages, including Java and C++.

Contact Numerical Algorithms Group, LTD, Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, United Kingdom, www.nag.co.uk.

SuSE 8.1

SuSE has released version 8.1 of its OS distribution, featuring improved ease of installation, security and stability. YaST2 (yet another setup tool) now autodetects and integrates the newest USB 2.0 and Firewire devices; it also has an expanded text mode. The improved X configuration tool, SaX2, can configure touchscreens, vertical LCDs, 3-D options and multihead systems. Other new features include a system profile manager, single-click switching between peripherals, improved wireless LAN support for laptops, CUPS as the default printing system, SuSE Firewall 2, OpenOffice 1.0, GNOME 2.0, KDE 3.0.3 and a lot of other new software.

Contact SuSE, 318 Harrison Street, Suite 301, Oakland, California 94607, 888-875-4689 (toll-free), www.suse.com.

Archive Index Issue Table of Contents

Advanced search