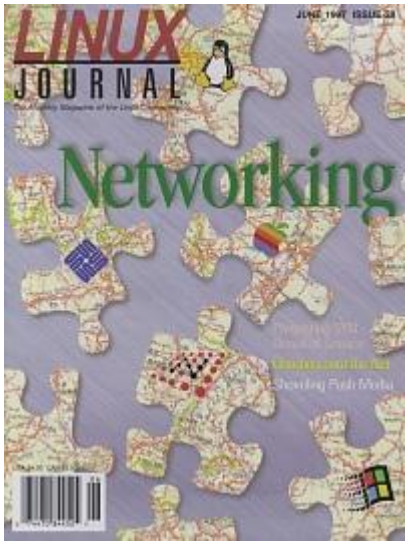
Linux Journal Issue #38/June 1997



Features

News & Articles

Reviews

WWWsmith

Who Is at the Door: The SYN Denial of Service

Douglas L. Stewart

P. Tobin Maginnis

Thomas Simpson

Issue #38, June 1997

How to survive the SYN attack on a TCP/IP protocol weakness.

Over the past few months, a denial of service attack, known as the “SYN Attack”, has become notorious. This attack can prevent access to your mail, WWW and other critical servers. The attack was first described in a paper by Robert Morris in 1985 and received little attention. It wasn't until *2600* magazine published source code to exploit this weakness in popular implementations of the TCP/IP protocol stack that this weakness grabbed the attention of Internet Service Providers. One provider, Public Access Networks Corporation of New York City, was attacked repeatedly last September, causing its mail and web servers to be unavailable to its users for extended periods of time. In this article we explain what SYN really is, why it's needed in TCP/IP, why the attack works and how to prevent it.

Introduction

The Internet works as well as it does because its data communication protocols (IP, TCP and UDP) evolved over a decade through major revisions and trial-and-error “adjustments”. As a result, the protocols have developed a legendary robustness that makes them difficult to defeat; however, these protocols were designed with the basic assumption that all network administrators can be trusted. Unfortunately, this is not true in today's Internet environment. Given the right kind of knowledge, virtually any PC can be configured so that a malicious individual, acting as a system or network administrator, can bring down any number of servers on the Internet.

One of these vulnerabilities is called the “SYN” (synchronous) attack, and it can affect anyone who places a server on the Internet. The SYN attack is a denial of service attack, blocking others from connecting to your server.

Network Layers

The Internet protocol stack utilizes three primary layers of the OSI model. The lowest layer is the physical layer, and it contains the physical wires, network host adapter(s) and adapter device driver(s). The next layer is the data link layer, whose job is to read a stream of bits off the network and assemble them into frames for the next higher layer.

The Internet Protocol (IP) or network layer is the next layer. It examines the incoming frames to determine if they are IP packets and, if not, it passes the frame onto another protocol stack (e.g., Novell) or discards the frame as nonsense. If it is an IP packet, the packet contents are further evaluated by the IP layer for a number of IP related activities such as Address Resolution Protocol (ARP) or Internet Control and Message Protocol (ICMP), which the connectionless **ping** and **traceroute** applications employ.

If the packet is not one of the above formats, its content continues to be evaluated as a Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) packet. If the packet contains a TCP header, it is posted to the next higher TCP layer. The verb “posted” is significant in that the packet is moved to another place for processing, and that processing will occur sometime in the future. In other words, it is at the IP-TCP boundary where information, driven by interrupts, “bubbles up” from the environment; it is at the IP-TCP boundary where information waits for processing based upon requests from programs that wish to communicate with the network. Therefore, the IP-TCP boundary contains a fixed amount of memory buffers allocated to network “activity” without the system really knowing what that activity is. It is at this boundary that the SYN attack works.

SYN Protocol by Analogy

Before discussing the third Internet layer and how TCP establishes a connection, perhaps it is better to begin with an analogy that illustrates a typical network problem and how TCP overcomes the problem in its daily routine.

Our analogy begins on a college campus with a studious student (SS) who has the misfortune of being placed in a “party” dorm. On a typical evening, SS is studying at his desk trying to master some dry material on data link protocols for his computer networks class. Someone knocks at his door. Upon opening the door, he gets hit with a water balloon from his rowdy neighbors. Using the

material from his network class, SS comes up with a solution to stop his pesky neighbors, yet still greet his invited visitors.

He decides on a “secret knock”—his friends announce themselves with a one to five knock code. SS hears the knock and goes to the door; however, he does not open it. Instead, he repeats the original knock and adds his own one to five knock code. Now the visitor knocks the next “sequence” of his code and repeats SS's knocks.

These knocking gymnastics are referred to as a three-way handshake (see Figure 1) in data communications lingo, and solve three common network problems. First, they allow two hosts to establish starting “sequence” numbers which are used by the receiver to re-order packets or reassemble datagrams. Second, they enable the host to identify duplicate packets that occur from re-transmissions which, in turn, are a result of delayed responses. Finally, if either computer were to initiate a connection with a third computer at the same time, then two orderly connections could result, without confusion.

Figure 1. The 3-Way Handshake

The Transport Station

Network traffic arrives at a given host and accumulates at the IP-TCP boundary, but nothing happens until a user-level process performs a request for network service through the transport station (TCP or UDP).

Most user-level Internet applications use a “virtual circuit” model for communication with web browsers such as Netscape or Lynx, FTP clients and Telnet clients. Steps in creating a connection or virtual circuit require the remote computer to request a “connect” which puts an IP packet in the local computer's IP-TCP boundary buffers. The local computer program requests a “listen”, then an “accept”. It is during these listen-connect-accept phases that TCP employs the three-way handshake to establish a virtual circuit.

Let's say there are two hosts, A and B, which exist on a network. A wishes to connect to B and issues a connect request. There are six bits defined in the TCP datagram header, two of which are the “SYN” (synchronize) and “ACK” (acknowledge) bits. The connect request datagram has the SYN bit set and the ACK bit cleared. When the process on host B receives the datagram, it accepts the sequence number, builds a reply datagram with B's separate sequence number plus host A's sequence number incremented by one, and the datagram is sent to A with the SYN and ACK bits on. Host A now has confirmation that B has provisionally accepted the connection, and it sends out the first data using the incremented sequence from its first datagram and returning B's incremented sequence number as an acknowledgment. The datagram now has

just the ACK bit set and when it is received by host B, the connection is established. (See Figure 2.)

Figure 2. Establishing a Connection Using 3-Way Handshake

The SYN Attack

Returning to the above analogy for a moment, we can see that the knock code is able to defeat SS's rambunctious neighbors, but what if they decide to knock once an hour or once every five minutes? What is our studious student to do? The knocks distract him from his homework, but if he ignores the knocks he misses any friends who come by. In other words, frequent knocks deny service to SS's friends.

The same is true at the IP-TCP boundary buffers. Once the host receives a SYN datagram and replies with an ACK datagram, how long does the host wait for the third part of the handshake? Unfortunately, current implementations wait forever.

Under normal circumstances, connections are established quickly, and so developers assumed that only a few buffers would be needed for all possible connections in the host. Under the 1.2.x Linux kernels, only 10 buffers are allocated.

To create a SYN attack, a program does not simply use the connect request; instead it opens a raw network connection directly and sends a burst of TCP SYN datagrams, ignoring any replies from the target host. The few buffers are now full and the target host is unable to establish any subsequent connections. Service has been denied to the target host. (See Figure 3.)

Figure 3. The SYN Attack

What makes this attack so insidious is that the attacker also inserts random IP source addresses in each datagram, thereby making it almost impossible for the remote host to trace the datagrams back to the real source.

A Case Study

An Internet Service Provider (ISP) closed a user's account because the user violated their acceptable use policy. This user now gets an account at a competing ISP and, armed with the latest issue of *2600*, dials up the new ISP using his PC running Linux. The user compiles the sample program given in *2600*, and runs it repeatedly against his old ISP's mail server and web server, filling up the connection queue on the ISP's servers. No one can receive mail or reach the ISP's web pages.

After restarting his web server several times, an administrator at the ISP runs **netstat** and notices a lot of the entries are flagged SYN_RECV. All of these entries are from random IP addresses. The administrator tries to ping several of the addresses, but they all fail to return any pings. The administrator then calls his network provider, a prominent National Service Provider (NSP), and requests help in tracking the attacks to the source. Unfortunately, the NSP is very busy maintaining its network, and doesn't have the resources to assist in such a search.

The ISP goes out of business.

Solutions

To lessen the severity of this attack, all providers should install the proper filters to prevent packages from leaving their network with forged source addresses, known as IP spoofing. This can be done by preventing packets that have a source address from outside your network from leaving your network.

Because the Linux kernel source code is under the GNU Public License (GPL), anyone with a copy of Linux is entitled to the source code. Having the source code, a user can apply a fix to his kernel and recompile it. If you were using a proprietary operating system, you would be at the mercy of your operating system vendor.

One of the easiest ways around this problem is to increase the size of the queue. This has been done in the 2.0.x kernels. If the queue is made large enough, it becomes more difficult for hosts with slow connections to the Internet (dial up, dynamic IP connections) to flood enough packets to prevent normal connections.

For your network servers to take advantage of the larger queue, they must be recompiled with a larger value as the backlog argument for the listen() function. **Sendmail** and **inetd** (found in NetKit-B) are two important programs that must be recompiled to “SYN-proof” your system.

A patch from Alan Cox implements random dropping of uncompleted connections, which prevents the buffers from filling, although the number of partially completed connections in the listen queue can increase. This same patch, which has yet to be integrated into the 2.0.x kernels as of patch level 27, also disallows a single class C from using up more than 30% of the queue. This last method prevents attacks from providers who have installed the source filters discussed above and from exploiters who do not use random source addresses.

The patch for the current kernel (2.0.29) can be obtained from http://www.dna.lth.se/~erics/linux.html. To apply it, download and unzip the patch into the /usr/src subdirectory and type

patch < tcp-syncookies-patch-1. When you run **make config** (or **menuconfig** or **xconfig**), you will see two additions under “Networking Options”. Just compile them into the kernel.

Other methods of protection have been suggested on various Internet forums, including creative firewalls that establish the TCP connection and then pass it on. Several companies are marketing commercial products based on these ideas. These solutions are not necessary for Linux users. Network solutions such as those are for users who don't have the option of compiling a fixed kernel.

Conclusion

The Internet is undergoing a massive scaling, and as a result, it is no longer possible to identify a given network administrator. While the Internet protocols were designed for unreliable networks, they were not designed for untrusted networks.

Although the SYN attack has proven very effective in denying service to important servers, the problem is well under control in the Linux world. The combination of a larger queue and the random drop technique makes your Linux-based system relatively immune to this attack.

Douglas L. Stewart works for Pencom Systems Administration and graduated from the University of Mississippi in December. Douglas can be reached via e-mail at douglas@pobox.com.

P. Tobin Maginnis is an Associate Professor of Computer Science at the University of Mississippi.

Thomas Simpson is a graduate student in Computer Science at the University of Mississippi.

Archive Index Issue Table of Contents

Advanced search

Network Management & Monitoring with Linux

David Guerrero

Issue #38, June 1997

Some handy tools for managing today's ubiquitous networks.

In today's world, where all the computing revolves around the concept of networking, the work for system administrators has become more and more overwhelming. It is the mission of maintaining the availability of resources such as routers, hubs, servers and every critical device in the network.

There are many reasons managers would like to monitor network devices: bandwidth utilization, operational state of links, bottlenecks, problems with the cabling or routing information distributed between its devices, etc. Monitoring network activity is also a good starting point for discovering security problems and misbehaviors.

In many cases, the network of an organization includes expensive links to remote networks (WAN) or the Internet, whose costs may be based on traffic volume. It's very important to maintain statistics of traffic going through these links. This is a very common task in Europe, where X.25 links are still very common. These links are charged on the basis of packets transmitted and received.

Other types of links, like Point to Point or Frame Relay, are usually charged on a flat rate. In these, the telco ensures a bandwidth that is important to monitor.

In the final part of this article we focus on a tool designed to monitor traffic in router interfaces, with a great graphical representation of this information. It can be easily modified to monitor other kinds of information.

What's SNMP?

The answer to all these needs is a protocol named Simple Network Management Protocol (SNMP). Designed in the '80s, SNMP's initial aim was to

integrate the management of different types of networks with a simple design that caused very little stress on the network.

SNMP operates at the application level using TCP/IP transport-level protocols so it can ignore the underlying network hardware. This means the management software uses IP, and so can control devices on any connected network—not just those attached to its physical network. This also has disadvantages: if the IP routing is not working correctly between two devices, it's impossible to reach the target to monitor or reconfigure it.

There are two main elements in the SNMP architecture: the agent and the manager. It's a client-server architecture, where the agent is the server and the manager is the client.

The agent is a program running in each of the monitored or managed nodes of the network. It provides an interface to all the items of their configuration. These items are stored in a data structure called a management information base (MIB), which we explain later. It's the server side, as long as it maintains the information being managed and waits for commands from the client.

The manager is the software that runs in the monitoring station of the network, and its role is contacting the different agents running in the network to poll for values of its internal data. It's the client side of the communication.

There is a special command in the SNMP command set called **trap** that permits an agent to send unsolicited data to the manager, to inform it of events, such as errors, shutdowns, etc.

In essence, SNMP is a very simple protocol as long as all the operations it performs deal with the fetch-and-store paradigm, and this allows for a small commands set. A manager can perform only two different operations on an agent: request or set the value of a variable in the MIB of the agent. These two operations are known as get-request and set-request. There's a command to respond to a get-request called get-response, which is used only by the agent.

The extensibility of the protocol is directly related to the capability of the MIB to store new items. If a manufacturer wants to add some new commands to a device such as a router, he must add the appropriate variables to its database (MIB).

Almost all manufacturers implement versions of SNMP agents in their devices —routers, hubs, operating systems, and so on. Linux is not an exception to this, and publicly available SNMP agents for Linux can be found on the Internet.

Dealing with Security

SNMP provides very little support for authentication schemes. It supports only a two-password scheme. The *public* allows managers to request the values of variables, and the *private* allows these values to be set. These passwords in SNMP are called *communities*. Every device connected to an SNMP-managed network must have these two communities configured. It is very common to have the public community set to “public” and the private community to “private”, but it's very important to change these values to reflect the security policy of your organization.

What's the MIB?

SNMP defines a separate standard for the data managed by the protocol. This standard defines the data maintained by a device in the network and what operations are allowed on it. The data is structured in a tree form, and there is a unique path to reach each variable. This structured tree is called the Management Information Base (MIB) and is documented in several RFCs.

The current version of the TCP/IP MIB is MIB-II and is defined in RFC-1213. It divides the information a TCP/IP device should maintain into eight categories (shown in Table 1), and each variable included in this information must fall in one of them.

Table 1

The MIB definition of a particular item also specifies the data type it can contain. Usually, items of an MIB can store single integers, but they can also contain strings or more complex structures, like tables. Items in an MIB are called objects. Objects are the leaf nodes of the MIB tree, but an object can have more than one instance—for example, a table object. To refer to the value contained in an object, you must add the number of the instance. When only one instance exists for an object, this is the **0** instance.

For example, the object **ifNumber** from category “interfaces” contains an integer with the number of interfaces present in this device, but the object **ipRoutingTable** from category “ip” contains the routing table of the device.

Remember to use the number of the instance to retrieve the value for an object. In this case, the number of interfaces present in a router can be viewed with the instance **ifNumber.0**.

In the case of a table object, you must use the index of the table as the last number to indicate a specific instance (row of the table).

There is another standard by which to define and identify MIB variables, called Structure of Management Information (SMI). SMI specifies MIB variables must be declared in an ISO formal language called ASN.1 that makes the form and contents of these variables unambiguous.

The ISO name space is within a global name space with other trees for other standards organizations. Within the ISO name space there is a specific tree for the MIB information. Within that MIB part of the tree are areas for objects from all protocols and applications so their information can be represented unambiguously.

Figure 1 shows the TCP/IP MIB name space is located just down the mgmt name space of the IAB. The hierarchy also specifies a number for each of the levels.

Figure 1. TCP/IP Organizational Tree

It's important to notice that most of the software needs the leading dot (root) to locate the object in the MIB. If you don't include the leading dot, it assumes a relative path from .iso.org.dod.internet.mgmt.mib-2.

This way the object **ifNumber** from category “interfaces” can be named:

```
.iso.org.dod.internet.mgmt.mib-2.interfaces.ifnumber
```

or its numerical equivalent:

```
.1.3.6.1.2.1.2.1
```

and the instance as:

```
.iso.org.dod.internet.mgmt.mibxi-2.interfaces.ifnumber.0
```

or its numerical equivalent:

```
.1.3.6.1.2.1.2.1.0
```

Additional MIBs can be added to this tree as vendors create them and publish the suitable RFCs.

What's the Future of SNMP?

A new specification called SNMPv2 is being actively developed. It addresses the lack of security of the actual protocol with mechanisms that focus on privacy, authentication and access control. It also allows more complex specification of variables and has some additional commands. The problem with SNMPv2 is it still is not a commonly accepted standard, unlike SNMPv1. It is not easy to find

SNMPv2 versions of the agents and software to take advantage of the new commands. Let's see what happens in the near future...

SNMP with Linux

One of the most popular SNMP packages is CMU-SNMP. Originally designed by Carnegie Mellon University, it has been ported to Linux by Juergen Schoenwaelder and Erik Schoenfelder. It's fully compliant with the SNMPv1 standard and includes some of the new proposed functionalities of SNMPv2.

The distribution contains some manager tools that permit, in a command line style, send requests to devices running SNMP agents. It also contains an SNMP agent program, designed to run under Linux, that provides managers running on the network (or the same system) information about the status of the interfaces, routing table, uptime, contact information, etc.

One very valuable add-on that comes with CMU-SNMP is a SNMP C-API, which lets programmers build more complex management tools based on the networking capabilities of the distribution.

The installation on a Linux system is easy, but a little different from the original CMU distribution. The distribution comes with precompiled binary versions of the manager tools, the daemon and the API library.

First of all, you must decide whether to get the binary or the source distribution. It's easy to locate the package on the Internet (check the resources sidebar). The binary distribution runs cleanly with the 2.0 kernel series and is ELF-based. We will explain how to install the binary distribution. It's a good practice to get binary distributions only from trusted sites to avoid viruses, Trojan-horse style attacks and other security problems.

Put the file cmu-snmp-linux-3.2-bin.tar.gz in the root directory (/) of your Linux system and decompress it with the command:

```
gunzip cmu-snmp-linux-3.2-bin.tar.gz
```

Then, untar the distribution to its final location with the command:

```
tar xvf cmu-snmp-linux-3.2-bin.tar
```

Now you will have all the utilities and libraries properly installed on your system, except the SNMP agent configuration file /etc/snmpd.conf. You can create it by running the script:

```
/tmp/cmu-snmp-linux-3.2/etc/installconf
```

with these options:

```
/tmp/cmu-snmp-linux-3.2/etc/installconf -mini <password>
```

where **password** is the public community you want to use. Now you can edit the newly installed configuration file /etc/snmpd.conf. In it, you can change the values for the UDP port used by the agent, the systemContact, systemLocation and systemName variables and the interface speed parameters for your network cards and PPP ports.

The most important management tools you get are:

- **/usr/bin/snmpget** A tool designed to ask for a concrete value in the MIB of an agent in the network (a router, a hub, etc.)
- **/usr/bin/snmpgetnext** It allows you to get the next object in an MIB tree without knowing its name.
- **/usr/bin/snmpset** A tool to set values in remote agents
- **/usr/bin/snmpwalk** Tool that requests a complete object or series of objects without having to specify the exact instance. It's useful for requesting table objects.
- **/usr/bin/snmpnetstat**
- **/usr/bin/snmptrapd** Daemon that listens for traps sent by agents
- **/usr/bin/snmptest** Interactive tool designed to demonstrate the capacities of the API.

The agent is located in the /usr/sbin/snmpd directory.

CMU-SNMP also installs an MIB file in /usr/lib/mib.txt. It's a good reference to search for information we can request from a device.

The agent must be run at startup time, and can be set up with this line in one of your system boot files (/etc/rc.d/rc.local, for example):

```
/usr/sbin/snmpd -f ; \
        echo 'starting snmpd'
```

Once you have the SNMP agent running for your Linux box, you can test it with one of the management tools, entering:

```
/usr/bin/snmpget -v 1 localhost \
        public interfaces.ifNumber.0
```

which will return the number of network interfaces configured in the system, and:

```
/usr/bin/snmpwalk -v 1 localhost \
         public system
```

will return all the values in the system subtree of the MIB. (See Figure 2 for the output of this command.)

The C-API is located in /lib/libsnmp.so.3.1.

You can check the related header files as follows:

- /usr/include/snmp/snmp.h
- /usr/include/snmp/snmp_impl.h
- /usr/include/snmp/asn1.h
- /usr/include/snmp/snmp_api.h

and more information in the man pages snmp_api(3) and variables(5).

There's also a Perl extension module to interface with the CMU C-API that easily integrates calls to this library in Perl scripts.

MRTG: Multi Router Traffic Grapher

MRTG is an advanced tool written by Tobias Oetiker and Dave Rand to graphically represent the data SNMP agents brings to SNMP managers. It generates nice HTML pages with GIF graphics about inbound and outbound traffic in network interfaces in *almost* real time. This abstracts the idea of dealing directly with objects of an MIB with a command line tool like CMU-SNMP. This is the simplest and most powerful tool to monitor my routers I have found on the Internet.

MRTG uses an SNMP implementation coded entirely in Perl, so there is no need to install other packages. The main program is written in C to speed up the logging process and the generation of GIF images. The graphics are generated with the help of the GD library from Thomas Boutell, author of the WWW FAQ.

One of the highlights of MRTG is its expandability and powerful configuration. It's very easy to monitor any SNMP variables instead of traffic, like error packets, system load, modem availability and others. It's even possible to import data from an external program to feed the data, so you can use it to monitor login sessions and other information not available through SNMP.

It comes with some tools to watch your router for interfaces, extract their characteristics and generate a base configuration file you can easily tweak to accommodate your needs.

Another interesting feature of MRTG is the amount of information it generates. It permits four levels of detail for each interface: traffic in the last 24 hours, the

last week, the last month and a yearly graphic. This allows you to gather information for statistical purposes. It maintains an accumulated database with all this information with the help of a consolidation algorithm that prevents the data in the logs from eating up your disk space.

It also generates a main page that contains the GIF images of the daily details of every interface of a router, which lets you have a complete idea of what's happening in your router with a simple look. You can see the main page and a detail page generated by MRTG in Figures 3 and 4.

Figure 4. Interface Detail Page

Let's see a basic installation procedure. First of all, you need the distribution of MRTG. At the time of this writing, the latest version was 2.1; check the URL in the references sidebar for the latest version.

A package you must install before compiling MRTG is the GD graphic library. The URL is in the references sidebar, too. The current version of GD is 1.2, and you shouldn't have any problems compiling and installing it. Simply run **make** in the directory you unpacked the distribution and a file called libgd.a will be generated. Copy this file to /usr/local/lib and all the .h files to the directory /usr/local/include/gd.

At this point you should have GD up and running. Now is the time to build the MRTG package. Unpack the distribution, and edit the Makefile, indicating where to find the GD libraries and header files, and the Perl 5.003 binary—usually /usr/bin/perl or /usr/local/bin/perl. This is done through the variables **GD_LIB**, **GD_INCLUDE** and **PERL**.

Build the main program by typing **make rateup**, and when the compilation finished, enter **make substitute** to include the correct PATH to the Perl interpreter within the set of Perl scripts that MRTG uses.

Copy the following files to the final destination of the binaries (for example, /usr/local/mrtg): BER.pm, SNMP_Session.pm, mrtg and rateup. You can also copy to this location the two configuration programs, indexmaker and cfgmaker.

Ensure that all the programs have the execution bit set. Now we're ready to build a simple configuration file. At this point you should have SNMP read access to your router. In a Cisco router, the configuration lines to allow this are the following:

```
access-list 99 permit 193.147.0.8
access-list 99 permit 193.147.0.9
```

```
access-list 99 permit 193.147.0.130
snmp-server community public RO 99
```

This allows read-only requests from the addresses specified in the access list 99 using “public” as a password (community). If you want to allow every node in the network Read Only (RO) access to the router, you can have a line like this one:

```
snmp-server community public RO
```

If you have another brand of router, check the manuals to determine how to allow SNMP access to them.

The **cfgmaker** script greatly simplifies the task of building the configuration file. All you have to do is run it with the following arguments:

```
cfgmaker <community>@<router-host-name or IP>
```

For example:

```
cfgmaker public@mec-router.rediris.es > mrtg.cfg
```

It will discover every interface in your router and write a section in the file with its specifications of numbers of interfaces, maximum speed, description, etc, with some HTML tags to include them in the detail page. It's possible to edit this HTML layout to suit your language, preferences, etc. You can see in Figure 5 the output for one of the interfaces of my router.

Router Interface Output Tree

Now you can run the **mrtg** program for the very first time. Simple execute:

```
./mrtg mrtg.cfg
```

If all goes well, it will contact your router, request some values, and generate some log files and several GIFs in the current directory. Don't worry about the complaints about the log and graphs not found, as this will happen only the first time. Remove the graphs and run the program again. The graph generated shows the traffic in the interval since you last ran the program. It also generates HTML pages for each interface.

Now it's time to instruct MRTG to run properly in your system. First, create a directory under the Document Root of your web server (assuming you run a web server on the same system) to accommodate the pages and graphs MRTG will generate each time it runs. Add this directory to the top of your configuration file with the directive WorkDir: /usr/local/web/mrtg (assuming that your Document Root is located in /usr/local/web). The next time MRTG

runs, it will create the logs and graphs in this directory, allowing you to access them via http://your_host.domain/mrtg.

Figure 3. Interface Main Page

Now, you would like to build a main page for all the interfaces like the one shown in Figure 3. This can't be accomplished with the indexmaker tool. Run:

```
indexmaker mrtg.cfg <router-name regexp>  >
        /usr/local/web/mrtg/index.html
```

It will generate an HTML page with the daily graphs of interfaces whose router name matches the previous regular expression and links to their single detail pages.

As you can imagine the MRTG program must be run on a regular basis to collect the data for each interval and generate the graphs periodically, in order to maintain the illusion of real-time monitoring. This is done through the following line in the crontab (assuming /usr/local/mrtg-bin as the mrtg program final destination):

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * *  \
        /usr/local/mrtg-bin/mrtg \
        /usr/local/mrtg-bin/mrtg.cfg  > \
        /dev/null 2>&1
```

In a Red Hat distribution, the correct line to append to the /etc/crontab file would be:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * *  root \
       /usr/local/mrtg-bin/mrtg \
       /usr/local/mrtg-bin/mrtg.cfg  >\
       /dev/null 2>&$
```

If everything is working fine, you can spend some time tuning your configuration and HTML index page. A good enhancement is to include in the <HEAD> section of the index page a <META .....> to force the browser to reload every 300 seconds to maintain the latest information on the screen.

Another enhancement you can include in your configuration file is the WriteExpire directive, which forces MRTG to create .meta files for each GIF and HTML page, eliminating unnecessary caching time by proxy servers and browsers. For this to work, you must also configure your Apache server (assuming you run the Apache web server) to read these .meta files and send the correct “Expire” headers with the **MetaDir** directive in the XXXX file.

You can look for additional directives in the example configuration from the distribution; it's very well documented. It's possible to alter all the layout of the images and pages generated by MRTG.

I hope you enjoy this program. If you do, send the authors a postcard; you can find their address on the MRTG home page.

Other programs

There is a similar program called Router-Stats, written by Iain Lea, the author of the well-known tin news reader. Router-Stats updates its graphics once a day and shows very interesting stats about hourly usage and other aspects. One problem with Router-Stats is it uses a lot of external programs to do its work (CMU-SNMP for SNMP tasks, GNUPLOT to draw the graphics, NetPBM to make some graphic conversions, and GIFTOOL to convert them to the final GIFs). You can check the URL for Router-Stats in the references sidebar.

There is another category of software that goes one step beyond in network management tasks and offers a complete solution for both monitoring and maintaining the distinct configuration of a whole network. This kind of solution permits us to draw a complex graphic representation of our network and browse through the nodes, checking specific items of the configuration and other interesting features.

At this level, we can talk about two commercial solutions broadly used: HP-OpenView from Hewlett-Packard and SunNet Manager from Sun. They provide a complete platform for managing all the resources of the network from great graphical interfaces. They also come with network discovery tools to find all the network's elements that have running SNMP agents and databases to store all the data gathered from the network for statistical purposes. One important feature of these environments is their ability to be integrated with other vendors' more specific products, like Cisco's CiscoWorks, that allows a network manager to maintain a database with all its router configuration and even monitor graphically the back panels of their routers and all their connections.

There are two drawbacks to these products: they are commercial and they have no ports to Linux. Of course, there are also public domain solutions for these tasks. One of the best packages I've found for this is Scotty. Scotty is a TCL-based package that allows you to implement site-specific network management software using high-level, string-based APIs. Its companion product, Tkined, is a network editor that provides extensions to build a complete framework, integrating some tools designed to discover IP networks, support the network layout process or troubleshoot IP networks using SNMP in combination with other standard tools (e.g., traceroute). Scotty also includes a graphical MIB browser to allow you to explore MIB information.

You can check the references listing for both commercial and public domain network management software pointers.

Conclusions

SNMP is a simple but powerful protocol that can help us monitor our resources with little stress to the network. It's possible the extensions being developed now will increase the complexity and capabilities of this tool but they will also increase the resources needed to implement them.

In this article, we have explored a couple of tools found on the Net. There are a lot of tools being developed each day. You can check the Usenet newsgroup comp.protocols.snmp for announcements of new software and MIBS.

Resources

David Guerrero is a system and network manager for the Spanish Ministry of Education and Culture. He has been in Linux since the .98plNN days, and now he's enjoying his new SPARC-Linux box. When not working, he likes to spend his time with his love Yolanda, trying to play music or going out with his “colegas”. He can be reached at david@mec.es.

Archive Index Issue Table of Contents

Advanced search

Ghosting onto the Net

Scott Steadman

Issue #38, June 1997

Communicating from the office to home using a Linux server and the Internet.

Background

Recently I got the urge to tinker with managing my network at home in order to get some experience with Unix and heterogeneous network management. I have three Windows boxes (two with Windows 95 and one with Windows 3.1) hooked up to a Linux server. I use the LinTel box as both a local file server and as a gateway linking my home network to the Internet.

The software I use to handle the file server tasks is SAMBA. My primary reference for setting up SAMBA was the excellent article on the subject in the July, 1996 issue of *Linux Journal*.

In picking a dial-up program, I kept two requirements in mind:

1. I didn't want to manually log on to my ISP each time I wanted access to the Internet.
2. I didn't want my LinTel box to call up my ISP on startup and then remain connected until I shut it down. I wanted to be considerate of my ISP's other clients by not monopolizing a phone line.

A program written by Eric Schenk, called **diald**, satisfied both these requirements. I use **diald** to connect to my ISP whenever I have traffic destined for the Internet. It also automatically disconnects from my ISP if there is no traffic for a specified interval.

I work for various companies with access to the Net, and while at work, I like to access my home Linux server through the Net from time to time—just in case I find something neat during a lunch break that I want to tinker with at home. So I set up my server to connect to the Net at various random intervals between 15 and 60 minutes, loiter around for five minutes and disconnect if there is no

traffic. While my server is connected I can download anything I wish. I call this process ghosting.

These are the steps I went through to get ghosting to work. Depending on whether you already have Linux installed and what flavor it is, you may be able to skip some steps.

Linux Installation

The first thing I did was acquire Red Hat 4.0 from Red Hat Software, http://www.redhat.com/. I had heard good things about Red Hat and liked their "Red Hat Package Manager" for handling software bug fixes and upgrades—it sure makes life easier. I installed Red Hat by following the directions given during the install process.

Next, I downloaded the latest version of the kernel available at that time, 2.0.29, from **sunsite.unc.edu**, and configured my new kernel using hardware specific settings.

Another necessity for ghosting is IP masquerading. I found three good sources of information on IP masquerading:

1. The most definitive is the IP-Masquerading Resource home page at http://www.wwonline.com/~achau/ipmasq/.
2. The IP-Masquerading Mini-HOWTO, probably available at your favorite Linux site on the Net.
3. The last is the IP masquerading article in the July, 1996 Issue of *Linux Journal*. I downloaded the latest IP masquerading patch for kernel 2.0.28 and higher from the IP-Masquerading Resource home page, and it worked fine with my 2.0.29 kernel. Again, all I had to do was follow the instructions to reconfigure the kernel using the **make menuconfig** method. Here are the pertinent settings for IP masquerading to work:
4. Under **Code Maturity Level Options**, turn on "Prompt for development and/or incomplete code/drivers". (The IP masquerading code is still considered alpha code.)
5. Under **Networking Options**, turn on "Network firewalls", "Network aliasing", "TCP/IP networking", "IP forwarding/gatewaying", "IP multicasting", "IP firewalling", "IP accounting", "IP masquerading (EXPERIMENTAL)" and "IP tunneling".

After configuring the rest of the kernel, I just continued following instructions to build it. I recommend doing a **make zdisk** and making sure the system boots fine from floppy before doing a **make zlilo**. That way the old kernel doesn't get accidently blown away. My **make** procedure is:

```
make dep
make config
make -j5 zdisk
make -j5 modules
make modules_install
```

I then reboot from the floppy and keep an eye on the startup information. With a successful reboot, go back into the Linux source directory and do a **make zlilo**. The **-j5** switch causes **make** to spawn up to five compiles simultaneously. This method of compilation speeds up the build process tremendously.

Setting Up the PPP Daemon

After installing Red Hat I set up the point-to-point protocol daemon (pppd); this allows my Linux server to communicate with the Internet. The ppp daemon came with the Red Hat package, and installs automatically when a networking package is selected.

First, I set up a configuration file named /etc/ppp/options, then created a chat script to tell the ppp daemon how to communicate with my ISP. The configuration file I used looks like this:

```
modem
/dev/cua0
38400
asyncmap 0
defaultroute
```

The man page for the ppp daemon explains these lines in detail. The default configuration file that comes with Red Hat should suit your purposes. The only line to be concerned about is **/dev/cua0**—this line tells the ppp daemon where to find your modem.

Before **pppd** can be used to communicate to the Internet, you have to dial and connect to your ISP. This usually involves a process called handshaking, implemented by a program called chat. A chat script sends the chat program the instructions for logging into your ISP. A chat script is basically a series of **wait** and **send** strings. Red Hat provides a network configuration tool that runs under X-Windows and can be used to create and test chat scripts. I had a chat script called /etc/sysconfig/network-scripts/chat-ppp0 (see Listing 1[footnote]). I symbolically linked this script into my /etc/ppp subdirectory using the following commands:

```
cd /etc/ppp
ln -s /etc/sysconfig/network-scripts/chat-ppp0
```

You will need to modify my chat script by changing the phone number, username and password responses to match your own. You may also need to modify the line **ppp default** depending on the requirements of your ISP—contact your ISP for that information.

Listing 1. Chat Script

Now, there are some things I want the system to do right after a successful connect to, or disconnect from, the Internet. Fortunately, pppd has a couple of features that make this easy. When the ppp link comes up, the daemon checks for the existence of a script called **/etc/ppp/ip-up**. If this script exists, ppp daemon invokes it with the specified connection parameters. My version of this script appears in Listing 2—notice the comments at the top of the script indicate the parameters **pppd** passes to the script.

Listing 2. /etc/ppp/ip-up Script

When the ppp link goes down, the ppp daemon checks for the existence of a file called **/etc/ppp/ip-down**. If this file exists, it is invoked when the ppp link is terminated. The contents of my script are shown in Listing 3. This script mainly does some cleanup—undoing what I did in the **ip-up** script.

Listing 3. /etc/ppp/ip-down Script

Setting up the Dialer Daemon

Next, I acquired and set up the dialer daemon, **diald**. This handy-dandy piece of software waits until it sees an IP packet destined for the Internet and, if the ppp connection is not up, automatically starts the ppp daemon, which then connects to the Internet.

This package can be obtained from http://www.dna.lth.se/~erics/diald.html. A word of caution—the latest version of **diald** is 0.16. I am using 0.14. I've tried 0.15, but it had problems reconnecting once I terminated a connection. I have not had time to test out version 0.16. Version 0.14 works just fine for me. If you are interested in upgrading to the latest and greatest **diald**, send me e-mail, and I'll let you know if it works now. I should have it tested by the time this article is published. Just follow the included instructions to build and install **diald**.

Listing 4: /etc/ppp/diald-up Script

Once I installed **diald**, I created some scripts to bring it up and down easily. The script to bring it up is called **/etc/ppp/diald-up** and appears in Listing 4 with plenty of comments.

Since this script is somewhat obscure, I will cover it in more detail. The **route** command is used to tell the network software how to get from your computer to other computers and networks. Normally there is a default route the network software uses when it can't find another suitable route in the routing

table. To view your routing table, use the **netstat -rn** command. For more information see the **netstat** man page.

The first command in Listing 4 removes the default route in order to make sure it is free for **diald** or the ppp daemon to use. This removal is necessary, since sometimes **diald** and ppp won't re-assign the default route if one is already assigned.

The second command starts the dialer daemon. (For more details refer to the **diald** man page.) To use this line in your script, you will need to change three items:

1. the communications device **/dev/cua0**
2. the local address **10.10.10.1**
3. the remote address **192.168.1.2**

If you have a fixed IP address, you'll also need to remove the **dynamic** switch line from the script.

The third, fourth and fifth commands are used to set up the firewall. These commands have to be run after the dialer daemon, because it does the masquerading from the network out to the Internet via the default route. Whenever a packet needs to leave via the default route, the dialer daemon senses it and makes a connection to the Internet using the ppp daemon.

I also have a script to shut down the dialer daemon gracefully. I call it **/etc/ppp/diald-down** and the source appears in Listing 5.

Listing 5: /etc/ppp/diald-down

The dialer daemon can be communicated with using a named pipe specified on the **diald** command line in the **diald-up** script. I use the recommended name **/etc/diald.fifo**. This named pipe allows you to change various parameters of the program while it is running and to gracefully exit the program without resorting to the **kill** command.

The first command in Listing 5 tells the dialer daemon to clean up and get out. The second command resets the default route back to the Ethernet card.

Testing the Dialer Daemon

To test the **diald** script, execute **tail -f /var/log/messages** in one virtual console, and in another type **ping 192.9.9.1** to ping sun.com. After typing the **ping** command, you can toggle back over to the first console and watch **diald** spit out status messages. These status messages tell you if **diald** dials your modem and

activates **pppd** correctly. If ppp appears to connect properly, you can toggle back over to the other console and see if the ping is returned. If not, don't panic—just break out of it using a Ctrl-C and try again. Sometimes packets get dropped when **diald** is switching the route from the slip interface to the ppp interface.

I used the IP address in the above commands on the assumption that you do not have a name server running on your machine. If you are interested in getting a name server up and running on your machine—something I recommend—a couple of good sources of information are the DNS HOWTO and the *Linux Network Administrators Guide* by Olaf Kirch.

Create an Appear Script

Next I created an **appear** script. The **appear** script causes **diald** to connect to the Internet, then sends an indication of where the server can be reached to the desired location. I created a script called **/etc/ppp/appear** to do the work. This script appears in Listing 6.

Listing 6: /etc/ppp/appear Script

Last, I added an entry to the **/etc/crontab** file. This file is used by the **cron** daemon to determine what to run when. (For more information on **cron** take a gander at the **cron** man page.) This is the line I added:

```
30 07 * * 1-5 root /etc/ppp/appear
```

This entry tells the cron daemon to start your **appear** script Monday through Friday at 7:30 AM. The **appear** script needs to be started this way only once per day; it will then restart itself whenever the time is right.

After completing all these steps, I was set up to ghost on and off the Internet, and if you've been following these steps, you will be ready too.

A Note about Windows 95 Configuration

If you decide, as I did, to hook up some WinTel boxes to your Linux server, here are some hints to get it up and running.

In the following examples, I am assuming your personal network is on the 192.168.1.* subnet, the Linux server is at 192.168.1.1 and your Win95 machine is at IP address 192.168.1.2.

Select the network icon in your Win95 Control panel. Then select the TCP/IP -> *network card* entry in the list. Click on properties, so that the properties window will appear, and do the following:

1. Under the **IP Address** tab, select “Specify an IP address”, and enter 192.168.1.2 in the IP Address field, also enter 255.255.255.0 in the “Subnet Mask” field.
2. Under the **Gateway** tab enter 192.168.1.1 in the “New gateway” field, and click the **Add** button. This tells Windows that the Linux server is the gateway.
3. Under the **DNS Configuration** tab select “Enable DNS”, and enter the host name for your machine in the “Host field”. Then enter the domain you use for your internal network.
4. If you have the DNS name server running on your Linux server, enter 192.168.1.1 in the “DNS Server Search Order” field and click **Add**. If you are going to use your ISP's name server, enter your ISP's name server IP address in this field instead.
5. In the “Domain Suffix Search Order” field, you can re-enter your internal domain and click the **Add** button.
6. Last, click on the **Okay** button. Windows will reboot and you will be set to go.

Conclusion

This setup has worked quite well for me. Every morning before I go to work I decide whether I want to be able to access my box from the office through the Internet. If I do, I just turn it on, and at 7:30 AM cron starts the **appear** script, and I'm off to the races.

There are some security issues to be aware of—once your server is on the Net, anyone can access it. To prevent people from being able to **telnet** to your server from anywhere, add the following line to your /etc/hosts.deny file:

```
ALL: ALL
```

This entry denies access to your box from everywhere—it is a good default. Now add the following entry to your /etc/hosts.allow file:

```
ALL: LOCAL myisp.net mywork.com
```

This entry allows you to connect only from systems on your local network, your ISP and your place of work. (For more information about these files, see the man page for hosts.allow.)

Scott Steadman (ss@stdmn.com) is a contract programmer who lives in Lawrenceville, Georgia with his lovely wife Kim and their two cats.

Archive Index Issue Table of Contents

Advanced search

Consistent Keyboard Configuration

John F. Bunch

Issue #38, June 1997

Eliminate inconsistent behavior from your keyboard by following the instructions in this article.

One of the convenient features of Linux is that the keyboard can be completely reconfigured to suit personal tastes. This feature can be a blessing or a curse when keys do not perform the same actions in all applications, but with a little work you can program any key to perform almost any task. Inconsistent keyboard behavior can be eliminated, and applications customized as desired.

In this article, you learn how to achieve consistent behavior for the **BACKSPACE**, **DELETE** and **ALT** keys. The Caps Lock key is switched with the left **CTRL** key to make the typing of control characters easier. The keys of the editing keypad are configured to perform as labeled. Function keys, and some keys of the numeric keypad, are programmed to perform arbitrary tasks. A shutdown key is also configured.

The keyboard configuration techniques will be demonstrated by a wide range of examples, one program at a time. Where practical, these techniques will be demonstrated for bash, less, Netscape, minicom and Emacs. Furthermore, the keyboard will be made to work as desired, regardless of whether the application is running in an xterm window, in a virtual console, or in an X11 window manager. However, it is assumed that the user has an IBM PC-compatible keyboard.

Definitions

An IBM PC-compatible keyboard is divided into five blocks of keys. The alphabetic keys and those surrounding them form the main keypad. To the immediate right of the main keypad are two small sets of keys. The upper six keys are the editing keys; the lower four keys are the arrow keys, which are also

called cursor control keys. At the far right is the numeric keypad. Finally, the function keys stretch in a single line across the top of the keyboard.

When a function key or an arrow key is pressed, an escape sequence is normally transmitted by the key. An escape sequence is a string of a few characters, the first of which is an **ESCAPE** control code. The rest of the string is used to distinguish one key from another.

The VT100 family is a set line of text-only display terminals once manufactured by Digital Equipment Corporation. The VT100 has become the de facto standard for ASCII terminals. Its successor, the VT200, is compatible with the VT100 family. However, VT200s also have a row of twenty function keys and six editing keys that the VT100s did not have. The VT200 keyboard is somewhat similar to the IBM PC-style keyboard that is now in general use.

Keypress Events

A keypress event occurs when the user presses any key on the keyboard. The event passes through various software programs, eventually resulting in some sort of action (see Figure 1). Ideally, the same keypress will result in a similar action in all programs, thus reducing the user's confusion.

For example, suppose a user is running Emacs inside an xterm under the X Window System and presses the up arrow key. Referring again to Figure 1, the up arrow is pressed, resulting in an event keycode of 98 being generated, which uniquely identifies that key. The X Window System translates this keycode into the "Up" keysym, which is received by the xterm. The xterm then translates the "Up" keysym into the three-character escape sequence "\\eOA" ("\\e" represents the ASCII ESCAPE control code). Emacs receives "\\eOA" as a series of three input events, which are then translated by the function-key-map into the vector [up]. The vector [up] passes unchanged through the key-translation-map. Finally, the global-map maps [up] to the previous-line command, which moves the cursor up one line in the buffer.

Figure 1. Keypress Event Flowchart

BACKSPACE, DELETE and ALT

One way to approach configuring the **BACKSPACE** and **DELETE** keys is to let them act as they do in DOS and MS Windows. That is, **BACKSPACE** erases the character to the left of the cursor, and **DELETE** erases the character under the cursor. This is a very convenient arrangement.

It is also a good idea to decide what control codes should be transmitted when these keys are pressed. Based on the key labels, let the **BACKSPACE** key

transmit a **BACKSPACE** (ASCII code 8, 010 in octal, 0x08 in hexadecimal, **CTRL-H**), and let the **DELETE** key transmit a **DELETE** (ASCII code 127, 177 in octal, 7F in hexadecimal). Simple solutions are usually the best.

The ALT key, which acts somewhat like a shift key, can be configured in one of two ways. Either it can set the eighth bit of the key being pressed with it, or it can cause an **ESCAPE** to be transmitted just before the key being pressed with it. Both methods will be used at different times. Sometimes it is simpler to configure the **ALT** key to transmit an **ESCAPE**. For example, holding down **ALT** while pressing A would cause the string "\\eA" to be transmitted. However, at other times it will be configured to set the eighth bit of the character being pressed, thus adding 128 to the ASCII code of that character. Meta is a synonym for **ALT**.

Linux Kernel

First, the kernel's keyboard translation tables will be redefined. Since these tables are not used directly by the X Window System, use one of the virtual consoles, not a window manager. Log in as root or use **su**. Different distributions of Linux may load the translation tables in different ways. To determine which keyboard translation table is in use, type:

```
# find /etc -type f | xargs grep loadkeys
```

You should see output something like:

```
/etc/init.d/boot:  loadkeys \
        /usr/lib/kbd/keytables/us.map
```

which would indicate that the U.S. translation table is in use.

Assuming the U.S. translation table is in use, enter the following commands to make a copy of it:

```
# cd /usr/lib/kbd/keytables
# cp us.map custom.map
```

The format of keyboard table files is given in keytables(5).

Now edit custom.map using any text editor. Find the following lines:

```
keycode 14 = Delete           Delete
        alt     keycode  14 = Meta_Delete
```

These lines specify that when *keynumber* 14 is pressed (the **BACKSPACE** key), send a **DELETE** to the system, and when **ALT-BACKSPACE** is pressed, send a Meta_**DELETE**. To find out the keynumber of any key, use the **showkey** command.

To make the **BACKSPACE** key conform to the design decisions, change these lines to read:

```
keycode  14 = BackSpace        BackSpace
         alt     keycode  14 = Meta_BackSpace
```

However, a delete key is also needed, so replace the following line:

```
keycode 111 = Remove
```

with these two lines:

```
keycode 111 = Delete           Delete
        alt             keycode 111 = Meta_Delete
```

Keynumber 111 is the DELETE key on the editing keypad, just below the INSERT key.

Now, to swap Caps_Lock with the left CTRL key, redefine keycodes 29 and 58 as follows:

```
keycode  29 = Caps_Lock    # Left Control key.
keycode  58 = Control      # Caps Lock key.
```

Configuring the numeric keypad presents a special challenge. In **us.map**, several of the keys transmit the same escape sequences as an editing key. This makes it impossible, for example, for a program to distinguish between the PAGE UP key on the editing keypad and the 9/Pg Up key on the numeric keypad. Furthermore, the NUM LOCK, /, *, - and + keys do not even transmit escape sequences.

To alleviate these problems, the virtual console's numeric keypad will be configured somewhat like a VT100 numeric keypad. Since xterm already emulates a VT102, this will save work by making the virtual consoles more compatible with xterm. Note that this technique could cause incompatibilities with software programs that expect the keys to behave as defined by us.map. If this becomes a problem, the keys can always be switched back.

To be able to configure the keys of the numeric keypad independently, they will have to be changed into function keys. Since the kernel supports up to 246 function keys, F1 through F246, this is not a problem. Redefine the following keycodes as shown:

```
keycode  55 = F112    # Numeric keypad *.
keycode  69 = F110    # NumLock.
keycode  71 = F107    # Numeric keypad 7.
keycode  72 = F108    # Numeric keypad 8.
keycode  73 = F109    # Numeric keypad 9.
keycode  74 = F113    # Numeric keypad -.
keycode  75 = F104    # Numeric keypad 4.
keycode  76 = F105    # Numeric keypad 5.
keycode  77 = F106    # Numeric keypad 6.
```

```
keycode  78 = F114    # Numeric keypad +.
keycode  79 = F101    # Numeric keypad 1.
keycode  80 = F102    # Numeric keypad 2.
keycode  81 = F103    # Numeric keypad 3.
keycode  82 = F100    # Numeric keypad 0.
keycode  83 = F116    # Numeric keypad ..
keycode  96 = F115    # Numeric keypad Enter.
keycode  98 = F111    # Numeric keypad /.
```

Furthermore, it is necessary to define the escape sequences that these keys transmit, so add these lines to the end of the file:

```
string F100 = "\\033Op"
string F101 = "\\033Oq"
string F102 = "\\033Or"
string F103 = "\\033Os"
string F104 = "\\033Ot"
string F105 = "\\033Ou"
string F106 = "\\033Ov"
string F107 = "\\033Ow"
string F108 = "\\033Ox"
string F109 = "\\033Oy"
string F110 = "\\033OP"
string F111 = "\\033Oo"
string F112 = "\\033Oj"
string F113 = "\\033Om"
string F114 = "\\033Ok"
string F115 = "\\033OM"
string F116 = "\\033On"
```

"\\033" is the octal representation of **ESCAPE**. These are the same escape sequences that will be transmitted by these keys when running xterm, after following the remaining steps.

Although function keys F6 through F12 are compatible with xterm, F1 through F5 are not. To fix this, add these lines:

```
string F1 = "\\033[11~"
string F2 = "\\033[12~"
string F3 = "\\033[13~"
string F4 = "\\033[14~"
string F5 = "\\033[15~"
```

It is recommended that the following keycodes be defined so the keysyms (the names following the equals signs) will match the keycaps. However, this will not change the escape sequences transmitted, since these keysyms are only synonyms for the original keysyms:

```
keycode 102 = Home      # Was Find.
keycode 104 = PageUp    # Was Prior.
keycode 107 = Enc       # Was Select.
keycode 109 = PageDown  # Was Next.
```

One very nice feature is to be able to hold down the **ALT** key while using the arrow keys to pan within Emacs. Since **ALT**-Left and **ALT**-Right were previously used to switch virtual consoles, those functions will be remapped to **CTRL**-Left and **CTRL**-Right.

Change these two lines as shown:

```
# Ctrl-Left (was Alt)
control keycode 105 = Decr_Console
# Ctrl-Right (was Alt)
control keycode 106 = Incr_Console
```

And add the following lines:

```
alt     keycode 103 = F117   # Left Alt-Up Arrow
altgr   keycode 103 = F117   # Right Alt-Up Arrow
alt     keycode 105 = F120   # Left Alt-Left Arrow
altgr   keycode 105 = F120   # Right Alt-Left Arrow
alt     keycode 106 = F119   # Left Alt-Right Arrow
altgr   keycode 106 = F119   # Right Alt-Right Arrow
alt     keycode 108 = F118   # Left Alt-Down Arrow
altgr   keycode 108 = F118   # Right Alt-Down Arrow
string F117 = "\\033\\033[A"   # Alt-Up Arrow
string F118 = "\\033\\033[B"   # Alt-Down Arrow
string F119 = "\\033\\033[C"   # Alt-Right Arrow
string F120 = "\\033\\033[D"   # Alt-Left Arrow
```

Note **ALT**-arrow transmits an **ESCAPE** followed by the normal escape sequence for the arrow key.

Of course, **CTRL-ALT-DELETE** will reboot Linux, but what if the user is finished for the day and wants a quick shutdown? To make **CTRL-ALT-H-END** shut down Linux, add the following lines:

```
# Numeric keypad End
control alt     keycode  79 = KeyboardSignal
control altgr   keycode  79 = KeyboardSignal
# Editing keypad End
control alt     keycode  107 = KeyboardSignal
control altgr   keycode  107 = KeyboardSignal
```

Then save the file, and edit **/etc/inittab**. Add or edit the following lines as shown:

```
# Action on special keypress (CTRL-ALT-END).
kb::kbrequest:/sbin/shutdown -h now
```

Save the file, and everything is ready to be tested. Type:

```
# loadkeys custom.map
```

and try the new keys. **DELETE** should act as **BACKSPACE** did before. Caps Lock has been switched with the left **CTRL** key. Exit all programs, then try **CTRL-ALT-END**. After shutdown, use the hardware reset button to reboot.

If the keyboard passed these basic tests, us.map can be replaced with custom.map. Regular users may encounter errors when running loadkeys, because it requires read access to /dev/console. Furthermore, different users using different maps could cause confusion. Therefore, it is suggested the new keymap be permanently installed on the system by root.

Warning: If these installation instructions are not followed correctly, it is possible to place the keyboard in an unusable state, forcing the user to reboot

from the installation floppies. Please take any necessary precautions before proceeding.

To install custom.map permanently, edit /etc/init.d/boot, or whichever boot script contains the loadkeys command, and add the following line to the top of the file:

```
custom_keys=/usr/lib/kbd/keytables/custom.map
```

Then replace the **loadkeys** command with:

```
if [ -f $custom_keys ]  # If custom keys exist,
then                            # then load them.
  loadkeys $custom_keys
else                            # Else use the regular keys.
  loadkeys /usr/lib/kbd/keytables/us.map
fi
```

This way, if **custom.map** somehow gets deleted, the keyboard will still work. Run this script to make sure it is correct. If it works, the new keymap will be automatically activated at the next system boot.

X Window System

Now start X as root. Change the directory to /etc/X11 and look at the file Xmodmap. Most likely, this file is empty, except for some comments. If not, rename it to Xmodmap.old, exit the window manager and restart X.

Starting with an empty Xmodmap is important, because when X comes up, it creates its keyboard modifier map and keymap table based on the kernel's current keyboard translation tables, which were just reconfigured. X then reads the Xmodmap file, which overrides the kernel. This could destroy some of the benefits of the work just completed on the kernel.

Since the kernel has already been reconfigured, the work of configuring X will be reduced by the work already done in creating custom.map. Specifically, **DELETE** and **BACKSPACE** will still transmit **DELETE** and **BACKSPACE**, because X got that information from the kernel.

Now enter the following command to replace Xmodmap with the current keymap table:

```
# xmodmap -pke > /etc/X11/Xmodmap
```

Edit Xmodmap. It appears similar to the file custom.map, but do not be misled. The keycodes are different. To find the keycode number for a given key, run **xev** from an xterm, put the mouse cursor inside the **xev** window, and press the key.

Look for the key's “keycode” in the output. For more information, see **xev(1)** and **xmodmap(1)**.

Finally, to switch the Caps Lock and the left **CTRL** key, add these lines to the end of the file:

```
! Recreate the Lock and Control modifier maps.
clear Lock
clear Control
add Lock = Caps_Lock
add Control = Control_L Control_R
```

To be able to configure keys on the numeric keypad, they must be given an appropriate *keysym* following the equals sign. Therefore, change the following lines as shown:

```
keycode  63 = KP_Multiply
keycode  77 = KP_F1
keycode  79 = KP_7
keycode  80 = KP_8
keycode  81 = KP_9
keycode  82 = KP_Subtract
keycode  83 = KP_4
keycode  84 = KP_5
keycode  85 = KP_6
keycode  86 = KP_Add
keycode  87 = KP_1
keycode  88 = KP_2
keycode  89 = KP_3
keycode  90 = KP_0
keycode  91 = KP_Decimal
keycode 108 = KP_Enter
keycode 112 = KP_Divide
```

Keycodes 99 and 105 are assigned to the keysyms Prior and Next, respectively. Just as was done in custom.map, change them to the following keysyms, which are functionally equivalent, but match the keycaps:

```
keycode  99 = Page_Up
keycode 105 = Page_Down
```

Now save the file, then type:

```
# xmodmap /etc/X11/Xmodmap
```

to activate the new definitions. If your keyboard will not work at all, use **CTRL-ALT-BACKSPACE** to exit X. Then use a virtual console to fix the problem.

Xterm

The xterm program receives keysyms from X and converts them into characters and into escape sequences. Since xterm emulates a VT102 terminal, it requires little configuration work.

Edit the /etc/X11/Xresources file, and add the following line:

```
XTerm*ttyModes: erase ^H
```

This automatically makes **BACKSPACE** (**CTRL-H**) delete characters to the left of the cursor. It is the same as if the user typed:

```
$ stty erase ^H
```

every time an xterm was started. Also add this line:

```
XTerm*appkeypadDefault: true
```

This will cause the keys of the numeric keypad to transmit their escape sequences instead of numbers, operators, etc. It is the same as if the user pressed **CTRL**--MiddleButton and selected “Enable Application Keypad” in every xterm window.

Unfortunately, Emacs normally resets the numeric keypad when it exits, so the keys no longer transmit escape sequences. To correct this behavior, the “rmkx” capability must be removed from the terminfo(5) database:

```
# cd /etc/terminfo/x
# infocmp xterm > xterm.txt
# emacs xterm.txt
```

Remove the “rmkx” entry, save the file, and exit. Then:

```
# tic xterm.txt
# rm xterm.txt
```

It is also necessary to tell xterm what the ALT-Arrow keys transmit, and what the **HOME** and **END** editing keys transmit. (The default escape sequences that xterm uses for **HOME** and **END** are unconventional.) Type the first seven lines below. Then cut and paste the remaining lines from the xterm(1) man page, underneath the heading, “The default bindings in the VT102 window are.”

```
XTerm*VT100*translations:\\
Alt <KeyPress> Up:string(0x1b) string(0x1b) string("OA")
Alt <KeyPress> Down:string(0x1b) string(0x1b) string("OB") \\n\\
Alt <KeyPress> Right:string(0x1b) string(0x1b) string("OC") \\n\\
Alt <KeyPress> Left:string(0x1b) string(0x1b) string("OD") \\n\\
<KeyPress> Home:string(0x1b) string("[1~") \\n\\
<KeyPress> End:string(0x1b) string("[4~") \\n\\
Shift <KeyPress> \
        Prior:scroll-back(1,halfpage) \\n\\
Shift <KeyPress> \
        Next:scroll-forw(1,halfpage) \\n\\
Shift <KeyPress> \
        Select:select-cursor-start() \\n\\
Shift <KeyPress> select-cursor-end(PRIMARY, \
        CUT_BUFFER0) \\n\\
Shift <KeyPress> Insert:insert-selection(PRIMARY,\
        CUT_BUFFER0) \\n\\
~Meta<KeyPress>:insert-seven-bit() \\n\\
Meta<KeyPress>:insert-eight-bit() \\n\\
!Ctrl <Btn1Down>:popup-menu(mainMenu) \\n\\
!Lock Ctrl <Btn1Down>:popup-menu(mainMenu) \\n\\
!Mod2 Ctrl <Btn1Down>:popup-menu(mainMenu) \\n\\
!Mod2 Lock Ctrl \
        <Btn1Down>:popup-menu(mainMenu) \\n\\
~Meta <Btn1Down>:select-start() \\n\\
~Meta <Btn1Motion>:select-extend() \\n\\
!Ctrl <Btn2Down>:popup-menu(vtMenu) \\n\\
!Lock Ctrl <Btn2Down>:popup-menu(vtMenu) \\n\\
```

```
!Mod2 Ctrl <Btn2Down>:popup-menu(vtMenu) \\n\\
!Mod2 Lock Ctrl \
        <Btn2Down>:popup-menu(vtMenu) \\n\\
~Ctrl ~Meta <Btn2Down>:ignore() \\n\\
~Ctrl ~Meta <Btn2Up>:insert-selection(PRIMARY, CUT_BUFFER0) \\n\\
!Ctrl <Btn3Down>:popup-menu(fontMenu) \\n\\
!Lock Ctrl <Btn3Down>:popup-menu(fontMenu) \\n\\
!Mod2 Ctrl <Btn3Down>:popup-menu(fontMenu) \\n\\
!Mod2 Lock Ctrl \
        <Btn3Down>:popup-menu(fontMenu) \\n\\
~Ctrl ~Meta <Btn3Down>:start-extend() \\n\\
~Meta <Btn3Motion>:select-extend() \\n\\
<BtnUp>>>>>>>>:select-end(PRIMARY, CUT_BUFFER0) \\n\\
<BtnDown>:bell(0)
```

Make sure each line ends exactly as shown, and there is not even a blank space after the last backslash on each line. Then change “Prior” to “Page_Up,” change “Next” to “Page_Down,” and change “Select” to “End” to match the keycaps.

ALT-Arrow transmits an **ESCAPE** (0x1b in hexadecimal) followed by the escape sequence for the arrow key, but **ALT**-<Keypress> sets the eighth bit of that key (examine the line that begins with “Meta<KeyPress>”). Also, the escape sequences for the ALT-arrow keys are not the same as were used in custom.map. For a full explanation of this file's format, see the **RESOURCES** heading under X(1). Save the file. Type:

```
# xrdb /etc/X11/Xresources
```

to activate the definitions. The BACKSPACE key should now work properly the next time an xterm is started.

Bash

Create a new file, /etc/inputrc for system-wide use or ~/.inputrc for personal use. This will be a readline startup file (see bash(1)).

First, make the **DELETE** key delete characters under the cursor, and make **HOME** and **END** work by adding:

```
DEL: delete-char
# Home.
"\\e[1~": beginning-of-line
# End.
"\\e[4~": end-of-line"
```

DEL is a special symbol **bash** understands to be **DELETE**. For the **HOME** and **END** keys, their escape sequences are given in quotation marks, followed by a colon, and then the command.

To determine the escape sequence transmitted by a key, look it up in custom.map. Alternately, bring up Emacs, then type **CTRL-Q** followed by the key. That will usually insert the escape sequence into the buffer. The ESCAPE character will appear as **^[**, i.e., control-left bracket.

The readline commands shown thus far are taken from the bash man page. If you would like to program a key to enter a regular command when pressed, put the command in quotation marks. For example, to make the 1 key on the numeric keypad list the current directory, add these lines:

```
# KP_1
"\\eOq": "ls\\C-m"
```

The **\\C-m** is a **CTRL-M**, or a carriage return, which, as we know, is necessary to execute the command.

Other keys may be programmed as desired. Save the file.

If a system-wide /etc/inputrc file was created, add the following line to /etc/profile:

```
export INPUTRC=/etc/inputrc
```

Then type:

```
# . /etc/profile
```

This will override the default initialization file, ~/.inputrc.

Regardless of where the file was created, it may now be activated by typing **CTRL-XCTRL-R**, which is normally bound to the readline command “re-read-init-file.”

Less

Create a new file, /etc/lesskey, for system-wide use. This will be a lesskey(1) input file. Type these lines exactly as shown and then save the file:

```
#line-edit
\\177    delete
^H      backspace
\\e[1~   home
\\e[4~   end
```

The control line, **#line-edit**, introduces bindings for line-editing commands. **\\177** is the octal representation of **DELETE**, and **^H** represents **CTRL-H**, which is a **BACKSPACE**. The escape sequences are those transmitted by the **HOME** and **END** keys.

Compile the file as follows:

```
# lesskey -o /etc/less /etc/lesskey
```

Add the following line to /etc/profile:

```
export LESS="-k/etc/less$"
```

Then type:

```
# . /etc/profile
```

The next time less is executed, it will read the key definitions from /etc/less. When entering a command line at the bottom of the screen (for example, the pattern for a search command), these key definitions will be active.

If a user wants to add personal key definitions to less, they may be placed in a file, say ~/lesskey, then compiled as follows:

```
# lesskey -o ~/.less ~/lesskey
```

These personal key definitions will be activated in addition to those defined system-wide.

Netscape and Minicom

Since Netscape is from the DOS/Windows world, it requires no special configuration. The designers made **BACKSPACE**, **DELETE**, **HOME**, **END**, **PAGE UP** and **PAGE DOWN** work just as expected. It simply shows that somebody did a good job, and it lends credibility to this design.

If minicom is used, then as root, enter the command:

```
# minicom -s
```

This will activate the configuration menu for minicom. Select “Screen and keyboard” from the menu. Type “A,” then press the space bar. Use “B” to determine what **BACKSPACE** should transmit. Since most remote systems will use **DELETE** to delete to the left, “DEL” is probably the best choice here. If this creates a problem, change it to BS. Press **ESC** when done.

Back at the configuration menu, select “Save setup as dfl,” then “Exit from Minicom.”

Emacs

Emacs can be executed in at least three different environments: in a virtual console, in an xterm, or in an X11 window manager. To run Emacs within an xterm, the **-nw** option must be used. Regardless of the environment, the keys will be made to perform identical functions.

Emacs contains a series of many keymaps that translate input events into other input events and eventually into commands. Three of these keymaps are

diagrammed in Figure 1. The ellipses indicate the omission of keymaps beyond the scope of this article.

As root, change directory to /usr/lib/emacs/XX.XX/lisp/term, where XX.XX is the version of Emacs installed. This directory contains initialization files for several types of terminals. Read the README file for more information.

Make a backup of xterm.el, then edit this file to look like Listing 1 This file maps the escape sequences sent by the keys onto Emacs vectors of length 1. A vector is a general-purpose Emacs array, and vector values are enclosed in brackets, e.g.,.

When Emacs is executed in an xterm or a virtual console, it reads this file. The function-key-map is then configured such that when any of these keys are pressed, Emacs automatically translates the escape sequences into vectors. This makes it easier to configure Emacs, not only because all of the keys can be referred to by their names, but also because the programmer never needs to remember the escape sequences after this file has been created. Furthermore, these vectors have the same names as the X Window keysyms, thereby providing consistency among all three execution environments.

The **PAGE UP/DOWN** keys are now referred to as prior and next. These definitions provide support for some of the Emacs commands, such as calendar, which expect these keys to be available. Also, even though /etc/X11/Xmodmap defines these keys to transmit the Page_Up and Page_Down keysyms, Emacs sees them as prior and next when running under X.

Two sets of definitions are listed for the arrow keys, Reset or Normal and Set or Application. This is because the VT100 series of terminals has two modes of operation in which keys transmit different escape sequences. Notice that in /usr/lib/kbd/keytables/custom.map, **ALT-UP ARROW** 17) transmits **\\e\\e[A**, but in /etc/X11/Xresources, **ALT-UP ARROW** transmits **\\e\\eOA**. In both cases, these are an **ESCAPE** followed by the escape sequence transmitted by the **UP ARROW** key. One method of dealing with this problem is simply to define both sets of escape sequences, as is shown here.

The configuration file, xterm.el, needs be compiled for increased performance, as well as to overwrite the previously compiled version. After saving the file, enter the following command:

```
# emacs -batch -f batch-byte-compile xterm.el
```

Since this file is also designed for the virtual consoles, make links as follows:

```
# ln -s xterm.el con80x25.el
# ln -s xterm.elc con80x25.elc
trl
```

The .elc files, which are compiled, are the ones used by Emacs.

For running Emacs directly under X, some changes need to be made to x-win.el. Please make a backup copy, then find and change the following lines as shown:

```
(define-key function-key-map [backspace] [?\\b])
(define-key function-key-map [M-backspace] [?\\M-\\b])
(put 'backspace 'ascii-character 8)
```

Save the file. Compile it with:

```
# emacs -batch -f batch-byte-compile x-win.el
```

Emacs does not need to know about escape sequences when running directly under an X11 window manager, because it receives the keysyms directly and converts them into vectors automatically. However, since the design states that the **BACKSPACE** key transmits a **BACKSPACE**, not a **DELETE**, three lines had to be changed to make x-win.el comply. Note that M-backspace is short for Meta-backspace, which is a synonym for **ALT**-backspace.

Now it is finally time for the last step, assigning functions to the Emacs keys. Change the directory to /usr/lib/emacs/site-lisp, which is where local Emacs Lisp programs are stored.

Create a file named “ibmkey.el” to define the key bindings for an IBM PC-compatible keyboard. Enter as much of Listing 2, ibmkey.el, as desired.

The default Emacs key-translation-map is defined in /usr/lib/emacs/XX.XX/lisp/loaddefs.el. This file is included in the emacs-el (GNU Emacs LISP files) package of the Debian distribution. It normally maps the F1 key and the **HELP** key (if present) to **BACKSPACE**, because **BACKSPACE** (**CTRL-H**) is normally mapped to the Emacs help-command in the global-map. Refer to Figure 1.

However, to accommodate the design presented here, the key-translation-map will be redefined. Since Emacs is based on the assumption that **DELETE** is to be used to delete characters to the left of the cursor, and since the design presented here assumes that the **BACKSPACE** key will perform that function, the key-translation-map will be used to switch the **BACKSPACE** and the **DELETE** keys. After this is done, pressing the **BACKSPACE** key will cause a **DELETE** to come out of the key-translation-map, and pressing **DELETE** will cause a **BACKSPACE** to come out.

To reduce confusion, the variable “BACKSPACE” is defined to hold a **DELETE**, and the variable “DELETE” is defined to hold a **BACKSPACE**. This way, when **BACKSPACE** is pressed, the value of the variable “BACKSPACE” will come out of the key-translation-map, and when **DELETE** is pressed, the value of the variable “DELETE” will come out, thus eliminating the confusion caused by the switch.

Many function keys are defined as examples, using the global-map. Since **BACKSPACE** is no longer available for help, F1 is now mapped directly to that function, without the use of the key-translation-map. F4 is mapped to Undo. The **BACKSPACE** key and the editing keys are defined to perform their labeled functions, except for prior and next, which are already defined by Emacs (see loaddefs.el). While holding down ALT, the arrow keys will pan the text in all four directions. Finally, the keys of the numeric keypad are defined to perform various functions.

There are several sources of information for programming Emacs. Press F1 twice to access the extensive online help facility. Also, the *GNU Emacs Lisp Reference Manual* (800+ pages) is available at ftp://prep.ai.mit.edu/pub/gnu/elisp-manual-19-2.4.tar.gz for the truly serious Emacs Lisp programmer. This manual can also be ordered from the Free Software Foundation. The order forms are located in /usr/lib/emacs/XX.XX/etc/ORDERS*.

Now save, and then compile ibmkey.el:

```
# emacs -batch -f batch-byte-compile ibmkey.el
```

Finally, edit /usr/lib/emacs/site-lisp/site-start.el, and add this line:

```
(load "ibmkey")
```

This file will automatically load ibmkey.el (or ibmkey.elc, if ibmkey.el was compiled) when Emacs is started by anyone on the system. If personal key mappings are desired, place them in ~/.emacs.

Summary

The techniques for configuring and programming the keyboard for the Linux kernel, the X Window System, xterm, bash, less, Netscape, minicom and Emacs have been presented. Table 1. Configuration Files and Commands summarizes the configuration files and commands.

After following these instructions, the keyboard should be configured to act more like the keyboard the typical user is accustomed to. The editing keys will perform as labeled. The Caps Lock and left **CTRL** keys will be switched. Various other keys will perform useful functions. Furthermore, the basic techniques

discussed here can be applied to other computer programs which permit configuration of the keyboard.

John Bunch, bunch@ro.com, is a member of University Baptist Church in Huntsville, Alabama, where he sings in the choir. To pay his bills and those of his church, he works as a Software Consultant for Intergraph Corporation. He holds a B.S. in computer science from the University of Tennessee, Knoxville, and a M.S. in computer science from East Tennessee State University, Johnson City.

Archive Index Issue Table of Contents

Advanced search

Wabi: Caldera's Solution for Windows Applications

Dwight L. Johnson

Issue #38, June 1997

Wabi is a Sun Microsystems application that runs Windows 3.1, 3.11 and Windows for Workgroups on Unix.

- Caldera Wabi 2.2 for Linux
- Price: US$199
- Platforms: SPARC Platform Edition, Intel Platform Edition
- Ordering Information: http://www.caldera.com/, 800-850-7779 in the U.S. or 801-269-7012 internationally.
- Reviewer: Dwight William Johnson

I find it a bother to reboot my computer to run the Microsoft Windows applications I need, so I was very excited when Wabi showed up on my doorstep.

Wabi is a Sun Microsystems application that runs Windows 3.1, 3.11 and Windows for Workgroups on Unix. You must have one of these versions of Windows to install into the framework which Wabi sets up on your Linux system. Then you must install the Windows applications you wish to run into this environment. While Wabi itself is installed into /usr/opt, the Windows installation must be repeated in the home directory of each user who wishes to use Wabi. All of the directory structures created by these installations reside inside the Linux directory structure and are accessible from Linux.

The Linux version of Wabi 2.2 has been licensed and published by Caldera and retails for $199. Information about how to obtain Wabi is on the Caldera Web site: http://www.caldera.com/; you can also call them at 800-850-7779 and 801-269-7012 (international).

In addition to its own Caldera Network Desktop, Caldera claims Wabi also runs on the Red Hat, Debian and Slackware Linux distributions which have a 1.2.13

or later kernel and the X Window System (X11R6). A local or networked CD-ROM drive is required for installation. A minimum of 10MB free disk space plus space for Microsoft Windows and its applications is required on your hard drive.

Sun Microsystems has certified an impressive array of 16-bit Windows applications to run under Wabi 2.2 (see Sidebar).

Many other 16-bit Windows applications can also work under Wabi, they just have not been tested and certified by Sun Microsystems and Caldera. For example, I installed QuickBooks 3.0, and so far, it appears to work perfectly.

Wabi enables most of the capabilities of Windows:

- Cut, copy and paste between Windows applications
- Access DOS-formatted diskettes
- Run in enhanced mode
- Object linking and embedding (OLE) between Windows applications
- Dynamic data exchange (DDE) between Windows applications
- Network installation and use of applications
- Windows Sockets networking

And, because it is running in a Linux environment, you can also do the following:

- Cut, copy and paste between MS Windows applications and X Window System applications
- Access network file systems transparently
- Use on X terminals
- Run on one system, display on another system
- Run additional applications simultaneously on your desktop
- Share serial and parallel ports
- Support multiple simultaneous users on one system
- Access NetWare files and directories

Functions not supported include those requiring Microsoft Windows networking, special device drivers and DOS commands. In particular, Wabi does not support:

- MIDI or AVI
- Full NetWare IPX/SPX connectivity
- Shared Wabi Windows directories

Inside the attractive Caldera Wabi box, I found the *Wabi for Linux* CD-ROM and 180-page indexed user's guide. As a Linux user hardened to the perusal of man pages, READMEs and HOWTOs, I found the high quality of the illustrated *Wabi for Linux User's Guide* a refreshing change. Free software is a great concept, but I can easily justify spending a little money for a product with documentation that so clearly communicates what needs to be done.

Installing Wabi on my Linux box with Red Hat 4.0 and 2.0.27 kernel went without a hitch. When I first tried to bring it up, however, my system immediately froze. This sent me scurrying to find support, and I was very pleased with what I found. Caldera provides technical support by e-mail and also hosts two mailing lists, “caldera-users” and “wabi-caldera”. All three of these support sources got back to me quickly with solutions. In this case, I had to uninstall Metro-X 3.1.2, which does not work with Wabi, and replace it with XFree86-3.2. I also found I needed to temporarily disable the font server by calling Wabi with **wabi -fs**.

My next challenge was to install Windows: Wabi did not show me an **A** drive. A quick query to “wabi-caldera” disclosed I had to expand the permissions on my floppy drive with:

```
#chmod 666 /dev/fd0
```

Windows easily installed from floppy diskettes after this. Wabi also permits installation via network.

In my case, configuring Wabi was extremely easy. The defaults were adequate to access DOS files in the Wabi-created **C** drive. And the PostScript driver provided by Windows enabled me to immediately print to my Hewlett-Packard 5MP printer.

For more advanced configuration, Wabi provides the Wabi Configuration Manager, which is extremely easy to use and accessible from an icon inside the Control Panel. Here, the user can attach Windows diskettes, drives, COM ports and printers to their Linux equivalents both on a stand-alone Linux box or across a network. The user can also attach a DOS emulator to the Program Manager RUN command, the MS-DOS prompt and DOS applications launched from icons under Windows.

As the DOSEMU 0.63.1.66 packaged with Red Hat 4.0 does not allow parameters and Wabi requires them, I was not able to test this capability beyond a simple MS-DOS prompt. Using DOSEMU, I am able to access DOS through my virtual consoles and xterms quite to my satisfaction, so not being able to launch DOS applications from Windows icons is, for me, no great loss.

With that behind me, I noticed my keyboard did not work in the necessary way. For example, ALT-ESC and DEL did not work. By this time I had fortunately found an additional area of support—the Wabi area of the Caldera Web site at http://www.caldera.com/. I found documentation of the keyboard problem with XFree86-3.2 here, and I quickly implemented the fix—placing XkbDisable in my XF86Config and enabling .Xmodmap in my user directory.

One real nuisance is Wabi's creating sticky windows, which appear on every virtual X screen. As I was writing this review, some of the subscribers on the caldera-wabi list were working on a solution by launching Wabi under Xnest using the following script:

```
Xnest :1 -geometry 790x572+0+0 &
exec wabi -display :1 +fs
```

This works great to control the display of Wabi inside one window. I was unfortunately not able to use it, because Xnest does not inherit the keyboard controls from the original server but instead implements its own internal default, which is not the IBM keyboard needed for Microsoft Windows. So for now, when Wabi is running, it hogs all of the X screens.

Installing the four Windows applications I wanted to use—Aldus PageMaker 5.0, Aldus FreeHand 3.10, Excel 4.0 and QuickBooks 3.0—was quite routine, and they all appear to work quite well.

Wabi is fast. I don't notice any difference in speed between Windows running under Wabi and Windows running under DOS. Cut and paste between Windows and X Windows applications works as advertised. Hey, I really like this! Microsoft Windows has never been so much fun.

In concept, Wabi aims toward a seamless integration of Microsoft Windows with Linux. In practice, there are loose ends in the implementation. For example:

- The File Manager sets up a diskette for formatting, then gives an error. Formatting of diskettes must be done in Linux.
- The floppy drives must be world-writable in order for Wabi to function, thus compromising system security over a network.
- Wabi's sticky windows reduce the functionality of X while Wabi is running.
- Neither Wabi nor Microsoft Windows provides support for the popular 600-dpi printers. I had to settle for a 300-dpi PostScript driver. These are only minor nuisances, and workarounds are easy to come by. There are major factors, however, which limit Wabi's marketability:
- Not enough applications are supported.

- Support for only 8-bit displays limits Wabi's usefulness for layout and desktop publishing even though the applications do run.
- Microsoft is currently deploying 32-bit Windows 97, putting Wabi two generations behind.
- I feel the price point for Wabi is too high for the Linux market. A $49.95 Wabi would make a lot of money for Caldera. Wabi, at $199, may sit on the shelf until Linux is a mainstream operating system in corporate America.

If you want or need the versatility and convenience of running Microsoft Windows from your X desktop over your network, you don't mind paying for the privilege, and your expectation is not *too* great, I would suggest buying Wabi for Linux now. You won't be disappointed. It is a professionally conceived and executed product with many nice touches that will give you solid performance. If you like the idea of Wabi but want to run 32-bit applications, drive a 24-bit display, or use mostly applications not on the certified list, wait for Wabi 3.0. The Unix version is scheduled for release in July. Neither Sun nor Caldera has yet announced a Linux version. However, I will be very surprised if we don't see one. Let us hope Sun and Caldera will see the light and begin to price Wabi for the mass market Linux is rapidly becoming.

Dwight William Johnson has been working on and around computers since 1967. Linux has been his preferred platform since April 1996. He lives in a huge ranch home with his family and ten cats in Sequim on Washington State's Olympic Peninsula, where he has been known occasionally to plunk on one of the grand pianos in his living room or saw on one of the violins or even (God help us!) raise his voice in song. He can be reached via e-mail at djohnson@olympus.net.

Archive Index Issue Table of Contents

Advanced search

OSS/Linux Sound Driver

Jeff Tranter

Issue #38, June 1997

OSS/Linux is the version for Linux systems. It's not surprising that Linux is supported, as the code was based on, and is compatible with, the sound driver included in the Linux kernel.

- Manufacturer: 4Front Technologies
- Price: $20 US
- Reviewer: Jeff Tranter

Open Sound System (OSS) is a kernel-level sound card driver offered for a number of Unix-compatible operating systems by 4Front Technologies. OSS/Linux is the version for Linux systems. It's not surprising that Linux is supported, as the code was based on, and is compatible with, the sound driver included in the Linux kernel. The code is written by the same author, Hannu Savolainen, who continues to maintain the free version (OSS/Free).

Installation

I took advantage of 4Front Technologies' free trial offer to download and run the product for five days. If you decide to purchase it you receive a software license key that allows the software to run permanently.

I also downloaded the latest release of OSS/Linux (3.8-beta1-961205). I installed the software on a 166 MHz Pentium system with 32 megabytes of RAM running Red Hat Linux 4.0 and the 2.0.28 kernel. The sound card was a Creative Labs SoundBlaster 16 (Plug-and-Play version).

The package comes as a compressed tar file containing installation instructions and an install program. To install the software you have to run the **install** program as **root**. This invokes a curses-based user interface that helps you install and configure the driver for the sound card. I found it straightforward to use; my sound card and settings were all automatically detected. The install

program noticed that I had the standard Linux kernel sound driver module installed and informed me that I needed to disable it. Deleting the kernel-loadable module for sound did the trick.

Once installed, the sound driver is loaded using the supplied **soundon** script. The driver is implemented as several loadable kernel modules. A **soundoff** script unloads the driver, if desired. Once loaded you can then run any of the existing Linux sound applications.

Testing

I tried a number of sound applications to play and record sound samples, play MIDI and MOD files and operate the mixer. In general, everything operated identically to the standard sound driver included with the Linux kernel. It provides control of the DSP device, FM synthesizer, mixer and MIDI bus interface.

I did find one bug. Apparently the MIDI driver didn't clean up resources it used. If it was unloaded, loading again would report that the I/O port was in use. I reported this to Tech Support by e-mail and received a response back within a few hours.

SoftOSS

I downloaded the beta release of OSS/Linux in order to try a new feature, SoftOSS. To understand it, you need some background on how computers synthesize sound.

The first generation of computer sound cards used a technique called FM synthesis to generate computer music. This method is low in cost and requires little CPU power, but the music generated sounds like it was created by a computer and not a musical instrument.

Sampling techniques use a digital-to-analog converter to generate sound. This method can be very effective, as the sound of actual musical instruments can be digitized and used as samples. The disadvantage is that the digital-to-analog converters are expensive, so sound cards typically provide only one or two. Typical low end sound cards today provide both sampling and FM synthesis capability.

Wavetable synthesis cards combine the best of both techniques. They offer a number of channels (32 being typical). Each channel has its own digital-to-analog converter and dedicated memory on the sound card for storing sound samples. Hardware on the sound card does much of the work of mixing and playing the samples. The only disadvantage of wavetable cards is that the

additional hardware makes them considerably more expensive. One such card, the Gravis UltraSound, is supported by the standard Linux kernel sound driver.

SoftOSS provides software emulation of wavetable synthesis using only a low cost (non-wavetable) sound card. It does so by using the spare memory and processing power of the host computer. By implementing at the kernel level the same application programming interface as the Gravis Ultrasound card, it allows existing applications written for this card to work with low end sound cards. The only catch is that you need adequate memory and CPU power, but most systems today can meet this requirement (a 40MHz 486 with 16MB of RAM was the minimum needed for the pre-release software I reviewed).

I configured the SoftOSS driver using the setting for Pentium 100+ machines and tried using some Linux applications that previously required a Gravis Ultrasound wavetable sound card.

The gmod program is a player for music files in MOD format. It operated quite well. This was not particularly impressive, because there are a number of MOD players for non-wavetable cards (e.g., tracker) that work just as well with the standard kernel sound driver.

Playing MIDI files was more interesting. MIDI is a very popular music file format among musicians, but MIDI file players I've come across before used the FM synthesizer on my sound card. Using SoftOSS, the playmidi program, and sound sample “patch” files downloaded from the Internet, there was a dramatic improvement in sound quality as compared to using FM synthesis. It actually sounded like real musical instruments.

Note that at the time of writing, SoftOSS was still in a pre-release beta state, and may be an extra cost option when purchasing OSS/Linux. For a beta release it looked quite stable, the only problem being some minor glitches in the sound produced.

Evaluation

So for $20 you get a single machine license for a sound driver that is compatible with the one in the Linux kernel, a **play** command for playing sound files and the **soundon** and **soundoff** scripts for loading and unloading the driver. Included are two years of technical support and five years of software upgrades. Documentation is essentially some README files covering installation, but more information, including the sound application programming interface, is available on the vendor's web site.

What advantages does OSS/Linux have over the free sound driver in the kernel? Technical support is one advantage which may be important to you if you are

using the Linux sound driver in a commercial setting. The package does appear to be easier to install and configure, automatically detecting the card settings in most cases. It also offers support for a few more sound cards (e.g., the SoundBlaster AWE32) and has better Plug-and-Play support than the free driver. It seems to be fully compatible with any applications written for OSS/Free.

On the negative side, there is a cost involved, albeit a small one compared to most commercial software. Of more concern is the fact that you don't get the source code. This means that you can't fix bugs or modify or enhance the code yourself. It also makes the software more sensitive to different kernel versions. You may have to periodically download a new sound driver from the vendor's web site when you upgrade your kernel, although a “wrapper” program called **sndshield** that you can compile is provided to help get around these problems most of the time.

Improvements

When using the free kernel driver, I like the fact that it can be automatically loaded and unloaded on demand using **kerneld**. The OSS/Linux driver, while it uses modules, unfortunately doesn't seem to support this. Having to log in as **root** and run a command to load the driver is cumbersome, although most users would probably know enough to put it in a system startup script like rc.local.

The package doesn't come with any value-added sound applications (except a simple “play” program). The sound driver on its own isn't very useful. If it was to come bundled with some of the existing Linux sound applications, less experienced users could make better use of the driver right away. The way to package the product, in my opinion, would be as a CD-ROM that came with a number of sound applications, sound files, and programming documents, precompiled for Linux systems. This would turn it into a more useful product, especially for beginners.

Conclusions

If your sound card works fine with the free driver in the kernel and you aren't interested in SoftOSS, then you probably won't see this product as adding much value.

If you've fought unsuccessfully to get your sound card to work under Linux, particularly if it's a Plug-and-Play model, then you should give this product a try. You can get a free trial copy and it's well worth the cost. If you have a non-wavetable sound card and are intrigued by SoftOSS, then you may also be interested in this product.

Finally, the OSS product is also offered for a number of other Unix-compatible systems. For years Unix systems have had no clear standard for sound programming. 4Front Technologies is hoping that the OSS API will become a de facto standard for Unix systems. If successful, this will be an ironic example of the tail wagging the dog—Unix systems striving to be compatible with Linux. It also means that sound applications written for Linux will have the opportunity to run on a wider variety of Unix platforms, expanding their scope.

Resources

Jeff Tranter has been using and writing about Linux for about four and a half years. He's the author of the Linux Sound and CD-ROM HOWTO documents, and the book “Linux Multimedia Guide” published by O'Reilly and Associates. When not playing with computers, he enjoys ham radio, playing guitar and cross-country skiing. He can be reached via e-mail at jeff_tranter@pobox.com

Archive Index Issue Table of Contents

Advanced search

Linux in a Nutshell

Sid Wentworth

Issue #38, June 1997

This book is designed as a reference for Linux rather than as a teaching tool.

- Author: Jessica Perry Hekman
- Publisher: O'Reilly & Associates
- Price: $19.95
- ISBN: 1-56592-167-4
- Reviewer: Sid Wentworth

Since *Unix in a Nutshell* was published by O'Reilly years ago, it was only a matter of time before it was followed by *Linux in a Nutshell*. The appearance of this Linux version in bookstores affirms the growing popularity of Linux.

This book is designed as a reference for Linux rather than as a teaching tool. It contains a minimum of tutorial information. Covered topics include a subset of the Linux user commands, shells (including bash, csh and tcsh), Emacs, vi, ex, sed and gawk. There is a limited amount of information on programming and the programming commands. Finally, there are chapters covering the basics of systems administration and a listing of systems administration commands.

The section on user commands is about 115 pages. While some commands I know were missing from the book, there were many more commands, command options or usage information I learned from it. For example, there are six pages covering every facet of the **less** command.

Shell coverage is divided into three chapters: A short overview of shells in general is followed by a 30-page chapter on the bash shell and a 40-page chapter on the csh and tcsh shells. My only disappointment was the lack of information on the POSIX-compliant Korn shell available for Linux; ksh could easily have been incorporated into the bash chapter.

The chapters on the Linux editors are designed in the same manner as those devoted to the user commands, and get about equal coverage. That is, they provide a good reference source, while offering a minimum of tutorial information.

The inclusion of a chapter on awk (actually gawk) came as a nice surprise. With Perl becoming *the answer* for so many problems, it is nice to see coverage of a language that is easier to learn and powerful enough to solve a very large number of problems.

The programming chapter was so brief, I got the feeling O'Reilly included it only so they could publicize the book as covering programming. The best that can be said is that it is weak—you just can't cover programming in 35 pages. The chapter covers the basic commands well enough, but don't expect to find coverage of subjects such as RCS.

I found the chapters on systems administration to be very strong. The overview shows what commands do what in only 12 pages. Grouping commands into logical function areas (managing the kernel, mail, managing file systems, ...) makes it easy to find the one you need. Once you know which command you need, you can get more detail from the command usage section in which the commands are presented in alphabetical order.

Linux in a Nutshell is accurate. This single fact differentiates it from the current on-line or printed man pages. If the choice is between accurate and comprehensive, I'll pick accurate every time. Coverage of many important topics is done in just over 400 pages.

Is this book for you? If you are comfortable with the idea of working from the command line to communicate with your OS, *Linux in a Nutshell* is probably a good choice for you. It provides a reasonable subset of the available Linux commands plus enough related information (such as that on shells and editors) to turn you into a competent Linux user.

Sid Wentworth lives in Uzbekistan, where he divides his time between UUCP hacking and raising yaks.

Archive Index Issue Table of Contents

Advanced search

Programming with GNU Software

Randyl Britten

Issue #38, June 1997

The tools covered are Emacs, the C and C++ compilers gcc and g++, the gdb debugger, the make utility, the RCS revision control system and gprof, a utility for profiling and timing your programs.

- Authors: Mike Loukides and Andy Orem
- Publisher: O'Reilly & Associates, 1997
- Pages: 260
- ISBN: 1-56592-112-7
- Price: $39.95; includes CD-ROM
- Reviewer: Randyl Britten

Linux is built on twin pillars. The one we usually associate with Linux is the kernel developed by Linus Torvalds. And the other, the one we often take for granted, is the set of GNU tools that make up the portable software development system from the Free Software Foundation (FSF). *Programming with GNU Software* is a brief but thoroughly useful introduction to these tools and how to use them. The tools covered are Emacs, the C and C++ compilers gcc and g++, the gdb debugger, the **make** utility, the RCS revision control system and gprof, a utility for profiling and timing your programs. The CD-ROM contains the sources for these tools and compiled binaries for several popular Unix systems such as SunOS and Solaris for SPARC, HP-UX for HP9000, AIX for the RS/6000, Irix for Indigo, and DEC Unix for the Alpha.

The book and CD-ROM combination is not specifically intended for Linux users, but this is an important book for anyone who develops software for Unix or Linux. This book was described as “forthcoming” in an O'Reilly catalog over a year ago and I have been eagerly awaiting it since then. If you have a recent distribution of Linux, you probably already have most or all of the tools discussed here. Even so, nearly everything in the book, and many of the sources on the CD-ROM, can be useful to Linux beginners and experts alike. A

significant additional benefit is the discussion of cross-compilation and cross-development for a wide variety of platforms. If you are interested in portable software or porting applications, embedded systems, or just a good introduction to the GNU tools, this book is a good place to start.

The libraries on the CD-ROM are not the FSF versions governed by the LGPL (a General Public License for libraries). The libraries are rewritten versions from Cygnus Support, a company that ports and supports free software for commercial use, particularly in embedded systems. Using the Cygnus libraries avoids the more impractical restrictions of the LGPL for commercial embedded systems, and Cygnus requires only that you acknowledge use of their software. If you invent the world's greatest latte maker/bagel warmer, for example, and use the FSF libraries in the controller, you either have to make the source code available or provide a way for your customers to upgrade the libraries in case of a new release. If you have any questions about the legal issues, you should contact Cygnus Support. Check out their web site at http://www.cygnus.com/.

Using GNU Tools

After an introduction and discussion of free software philosophy in Chapter 1, Chapter 2 offers a brief introduction to the use of Unix and its traditional tools for software development. Then Chapter 3 gets into the heart of GNU software with Emacs. If you have been intimidated by the sheer mass of Emacs, you should find this chapter an easy way to get started.

The coverage in this book is not simply introductory, but also has depth where it counts. Even though I've used versions of Emacs for years I learned several new tricks in this chapter. For example, I like to use the original Kernighan & Ritchie indentation style in my C programs, and indent by four spaces with each press of the TAB key. Using an example in the book, I was able to add a customization to my default initialization file, .emacsrc, in my home directory. Now, whenever I edit a file that ends in .c, .cpp or .h, Emacs will automatically be in C-mode, and will indent my programs properly when I press the TAB key at the beginning of a line. Most of its features and all of its customizations are written in Emacs Lisp. Emacs itself is mostly an interpreter for this language and its many extensions. Also covered is how to run the compiler from within Emacs, so you can continue editing while compilation proceeds in the background.

The GNU software tools are integrated and work together in a variety of ways, and Emacs is at the heart of it. Here are some additional incentives for learning and using Emacs I've gleaned from the book:

1. bash—the default Linux shell—uses Emacs commands for basic command line and history editing.

2. info—the command often more helpful than man pages—is an Emacs-like hypertext application that uses a special set of commands for basic navigation plus many familiar Emacs commands.
3. gdb—a shell in its own right—uses Emacs commands for basic command line and history editing.

Because of the interoperability between Emacs, bash, gcc, GNU make, RCS and gdb, learning to use Emacs just makes good sense.

Compiling with gcc

Chapter 4 begins with an overview of the compilation process, including the preprocessor, translator/optimizer, assembler and linker, plus many other tools (such as the libraries) that take part in the software development process. There are many compiler options, optimization levels and intermediate file formats you can use. Like the rest of the book, this chapter does not attempt to be a comprehensive reference. Instead, it does a good job of discussing the most frequently used commands and options, and adds tips that even power users will appreciate. The chapter ends with an introduction to cross-compilers, and the requirements for building your own libraries in a cross-development environment. A table is included that outlines the large number of host and target systems supported by the Cygnus libraries, and the output formats such as a.out, COFF and ELF, that can be generated.

Chapter 5 continues with more details about using the C and C++ libraries, and what is needed to support the system interface to Unix- or Posix-like systems. So if you are interested in porting Linux to the latest 64-bit PDA or the new WebTV your aunt just bought on sale, this is the place to learn what it takes. Keep in mind that these two chapters are not about the C or C++ languages or how to write programs. Many other books are more useful as learning aids (see the Resources sidebar). Chapter 5 ends with a brief discussion of the library licensing issues.

Debugging with gdb

The GNU debugging tool, gdb, is an interactive shell with its own commands, history (previously executed commands) and editor (Emacs-like, of course). The basic idea is that you can control and examine the internal working of an executing process, and interact with its source code and variables. The coverage of gdb is extensive here, and I have not seen gdb covered with a good tutorial in any other reference. This coverage alone could be worth the price of the book.

Building Programs with make

The **make** utility is used to build programs from multiple sources and compiles only files in need of updating, based on the date stamps and dependencies for each file. It is fairly easy to write simple dependencies so that if an include file is changed, for example, only the files that use it are recompiled; automating these steps saves time when building a new executable program. The **make** utility has been around a long time and has become very sophisticated, and the GNU **make** is one of the most comprehensive. The coverage in Chapter 7 is brief, but is an excellent tutorial introduction that covers both basic and advanced features. For more in-depth coverage, see the O'Reilly book on **make**.

Managing Source with RCS

The RCS revision control system is a tool to manage the versions of a program as it evolves over time. GNU **make** is aware of RCS, and can automatically use the current revisions. Again, the coverage is brief but presents sufficient basics for you to start using them; further details are available in other works from O'Reilly and FSF.

Profiling with gprof

There are two tools for timing and profiling your programs: **time** and **gprof**. The **time** command is built into the bash shell and is similar to the **timex** command in other shells. It simply gives the elapsed execution time of a program as a whole, broken down by user and system, with a few additional system details. The **gprof** tool is a report generator that can provide detailed information on where your program is spending its time. The **gprof** utility can give either a one-dimensional profile or a two-dimensional accounting that follows the call graph of your program. The call **graph** starts with the **main()** function and has an execution breakdown for every function called that includes both the time spent and the number of times the function is called. It even handles recursive programs. Learning to use **gprof** is the best way to improve the performance of your programs. Coverage of this important tool has not been easily available elsewhere and is another reason why this book is a valuable resource.

Conclusion

What is particularly good about this book is the combination of an excellent tutorial style that makes it easy for you to get started, and depth that cuts to the important topics in each subject. Even if you are already experienced with C/C++ programming using Unix tools, you will find many useful tips. With only about 250 pages, this is brief coverage, and the one thing I might wish for is a more complete reference. For that we will have to turn elsewhere, such as the **info** pages and the references listed below. I'm sure this book will be a valuable

reference for me for some time. The authors, Mike Loukides and Andy Orem, are senior technical editors with O'Reilly and have done an excellent job that rises well above the average for software documentation.

Resources

Randyl Britten is a software developer in Seattle and has an MS in Computer Science from the University of Washington. He has been tinkering with Linux since about the time *LJ* put out its first issue. He also teaches C programming at UW Extension through the Web. You can find more at http://weber.u.washington.edu/~instudy, or send e-mail to britten@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Using mSQL in a Web-Based Production Environment

B. Scott Burkett

Issue #38, June 1997

This article provides insight for creating a full-fledged, web-based database application under Linux, using David Hughes' mSQL package and standard Unix tools.

Over the past few years, many companies have realized the benefits of using Linux to serve web content to the masses. The power of a freely available, feature-laden 32-bit operating system, coupled with a vast number of utilities and development tools, provides a cost-effective solution for implementing enterprise and publicly-available information servers.

While many organizations have championed Linux as a web server, few have taken advantage of perhaps one of the most interesting aspects of the Web: *dynamic content generation and delivery*. Think about it. Of all the web sites you visit on a regular basis, how many of them have static content? Not many. Many us go to Yahoo! each day to see “What's New on the Internet”. Many cruise over to catch the news on CNN throughout the day. These sites have dynamic content. If the listings on Yahoo! didn't change every day, how many of us would go back after the first visit?

To provide dynamic content to your cyberguests, you can use a variety of tools and methods. One of the more popular approaches is to integrate data repositories with the Web. Creating web-based applications that integrate with existing database pools seems to be the rage this year. This paradigm has led to some amazing third-party products such as Bluestone Software's Sapphire/Web (http://www.bluestone.com/) and Haht Software's HahtSite (http://www.haht.com/). These products provide full development environments for designing, creating and deploying web-based applications. Unfortunately, the majority of these products are not yet available for Linux (iBSC options ignored for the moment). However, there is an alternative.

You can retrofit a Linux-based web server to provide access to enterprise data in a very cost-effective manner. Third-party packages typically have an integrated development environment (IDE) to provide for seamless, somewhat painless development. This can be easily replaced by your favorite text/HTML editor. Third-party packages typically interface nicely with expensive, proprietary database platforms such as Oracle, Sybase and Informix. These database systems cost thousands of dollars, and generally require a seasoned database administrator (DBA) to operate efficiently. In our Linux model, we will employ David Hughes' mSQL engine, which costs a whopping $170 USD, and is a breeze to use. To fully implement such an approach, expect to spend no less than $10,000 on the software alone. The Linux/mSQL approach (including the cost of a Linux CD-ROM distribution, the mSQL engine and coffee) should cost around $250. Senior management has always had a love affair with saving money—show them the numbers. It sells itself, folks.

Requirements

In this article, the following assumptions are made:

1. You have a working, fully installed Linux server.
2. You have a functional HTTP server running (NCSA, CERN, Apache, etc.).
3. You have installed either BASH, pdksh or ksh93.
4. You have the standard Unix tools in place (awk, sed, Perl, etc.).

Obtaining the mSQL Package

The first item you need is the mSQL (mini Structured Query Language) engine itself. The mSQL package implements a relatively fast, lightweight database engine that uses a subset of the ANSI SQL standard to perform its operations. As of this writing, the current stable release is version 1.0.16, although the long awaited v2.0 release has been promised soon. It can be obtained via ftp at ftp://bond.edu.au/pub/Minerva/msql/. The official home of mSQL is at http://Hughes.com.au/.

Next, you need the **w3-msql** package, also written and distributed by David Hughes. This package provides the CGI (Common Gateway Interface) interface to the databases managed by mSQL. As of this writing, the current version of **w3-msql** is version 1.0, although 2.0 is in the works. It is available via ftp at ftp://bond.edu.au/pub/Minerva/w3-msql/.

Finally, the example scripts presented in this article are available via ftp at ftp://www.dcicorp.com/pub/unix/msqlweb/. Unless you are a typing enthusiast and are already familiar with mSQL, I recommend you snag the examples.

Installation and Compilation

Once you have obtained the distribution archive, move it to either a scratch directory or the base of your normal source tree. You can extract the package as follows:

```
gzip -d msql-1.0.16.tar.gz
tar xf msql-1.0.16.tar
```

To prepare for compilation, switch to the **./msql-1.0.16** directory and execute the following commands:

```
make target
cd targets/Linux*
 ./setup
```

You will be asked the following questions pertaining to the actual build of the package. Here are a few notes to guide you:

Top of install tree? While mSQL can be installed virtually anywhere on your system, you should use the default path, /usr/local/Minerva. It makes installing third-party add-ons easier.

Will this installation be running as root? This question is concerned primarily with the TCP port mSQL uses for network communication. If your distribution is running as root, the default TCP port is 1112; otherwise port 4333 is used. You can tailor these defaults in the **./common/site.h** header file. Also, take a look at the mSQL FAQ, available at the mSQL web site, which describes a number of other scenarios this setting affects.

Directory for PID file? Where do you keep your PID files? The default is **/var/adm**, which is fine for most folks.

At this point, the script will finish its tailoring process. Before you actually compile the package, you can perform several customizations by editing a few of the source files. The first, **./common/site.h**, contains such gems as selecting the German language over English for error reporting. Give it a quick glance and make sure you are comfortable with the settings. Another possible modification lies in the **./msql/msql_priv.h** file. I like to bolster my database limits a bit. At the top of this file are several values you can alter to suit your needs, including the maximum number of fields returned in a query, maximum number of network connections allowed, and the maximum length for field and table names. Feel free to modify these as you see fit. For the non-adventurous, the defaults should suffice.

To compile the package, simply execute the following command from the base source directory (**./targets/Linux***):

```
make all
```

Compilation on a Pentium-class machine generally takes a little over a minute. If there are no compiler errors, you can install the package by executing the following command:

```
make install
```

The system is installed in /usr/local/Minerva (or whatever you set the install directory to when you ran **setup**).

Compiling and installing the **w3-msql** utility is much simpler. After you obtain the distribution archive, extract it into your source or scratch directory as follows:

```
gzip -d w3-msql-1.0.tar.gz
tar xf w3-msql-1.0.tar
```

Change into the **w3-msql-1.0** directory, and remove the **-lsocket -lnls** assignment to the **make** variable **LIBS**. Linux does not require these libraries to be linked into the application. Run **make**, and you are in business. If the build was successful, simply copy the **w3-msql** binary image over to your web server's cgi-bin directory.

Testing

The mSQL server process needs to be invoked during your machine's startup procedure. Place a line similar to the following in your /etc/rc.d/rc.local file:

```
/usr/local/Minerva/bin/msqld&
```

For testing purposes, and to save you a reboot, execute the above command from the shell prompt. This gets the server process up and running, ready to handle your database requests.

To make sure your mSQL server has been installed properly, several test scripts are supplied with the mSQL distribution archive. Finally, make sure you take the time to look over the mSQL documentation.

Setting up mSQL for Use with the Web Project

As part of this article, a fully functional web-based database application is presented in its entirety. The purpose is to provide a framework for your own web endeavors, as well as to show off the ease with which you can construct these types of applications using standard Unix tools.

Our example focuses on the creation and interaction of a database to contain concert listings that contains several items of information for each concert. First, we concentrate on populating the database with a series of concert listings, then build some web-based queries.

Before we create the database schema itself, we need to determine the functional elements of the application. What information needs to be stored in the database? What types of queries will be offered to our web guests? Do we need the ability to allow additional concert entries to be added through the Web?

Our database stores the concert date, opening act, headlining act, location and ticket price. For now, we concentrate on simple queries such as looking up concerts by band name and location. We also assume the “add” page is either protected by HTTPD authentication or made available through an Intranet.

The following is the mSQL schema we used to create the concerts database:

```
create table notices (
        show_date               char(10),
        headliner               char(30),
        opening_act     char(30),
        location                char(30),
        ticket_price    char(10)
        )
```

To create the sample database and load it with initial data, execute the **mkconcerts** script located in the examples archive. For those of you who don't have ftp access, the data are shown in Listing 1.

Listing 1. mkconcerts Script

To verify that the data have been loaded properly, execute the **mkreport** script (see Listing 2. mkreport Script), which is also in the examples archive. This script simply dumps the contents of the database into a formatted table, called concert.listings, shown in Listing 3.concert.listings File

Web-Based Interaction with w3-msql

Now that the database is created and populated with test data, it's time to begin constructing HTML pages that interact with the database. (Keep in mind that this article is intended to supplement but not replace the documentation that ships with mSQL and w3-msql.)

w3-msql acts as an HTML “preprocesser” of sorts. It takes a standard HTML document and performs database actions based on embedded mSQL primitives as shown in Figure 1.

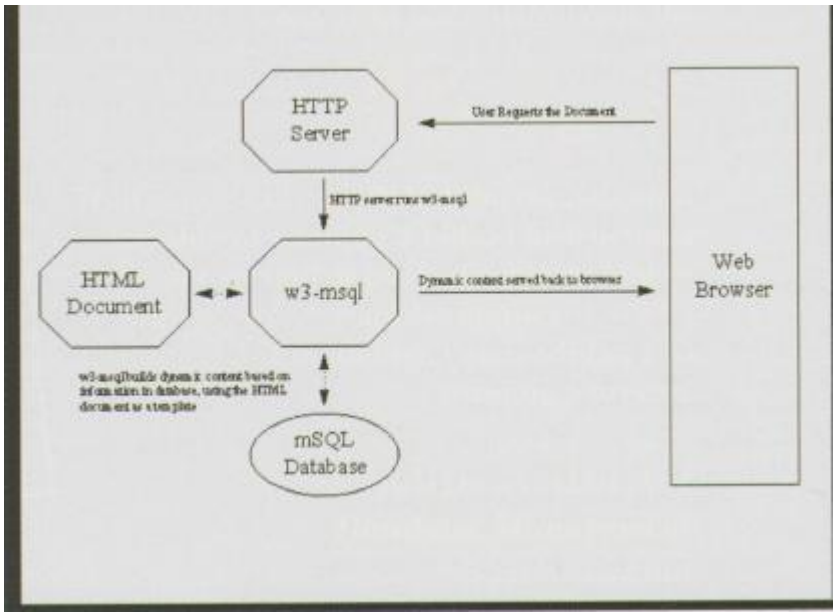


Figure 1. Process Flow

As a first example, consider the HTML document named ex1.html, shown in Listing 4. HTML Document with Embedded mSQL Commands, which demonstrates a simple link to a document containing embedded mSQL commands.

Note the calling procedure. The document name is placed immediately after the invocation to w3-msql (PATH_INFO). w3-msql takes this document, looks for any embedded mSQL commands and sends the appropriate output to the client. In this case, we are requesting the **query1.html** document, which contains the HTML shown in Listing 5. HTML Document. When selected, the output of the w3-msql link is shown in Figure 2.

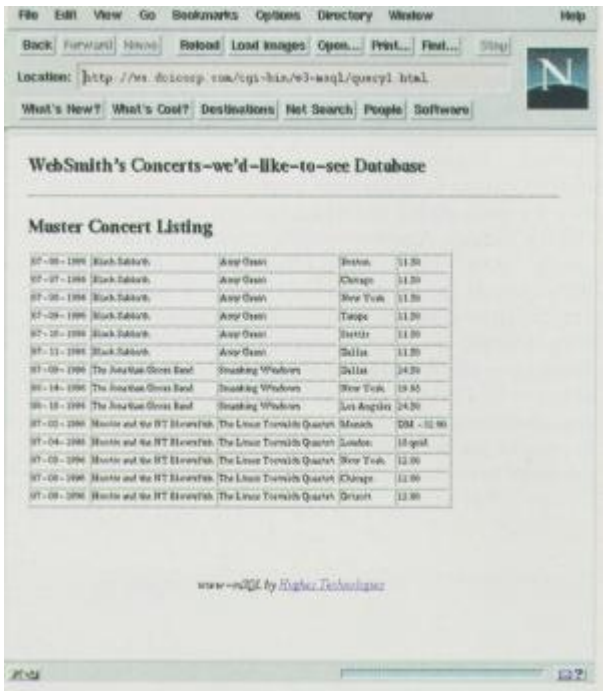


Figure 2. Output of ex1.html/query1.html

While the previous example is rudimentary, it demonstrates the ease of presenting information from an mSQL database. Let's continue with this example by expanding our queries. One nice feature to implement is the ability to produce a listing sorted by a specific field. Consider the following rewrite of our ex1.html file, which adds several hypertext links, each sorted on a different field. The file, called ex2.html, is shown in Listing 6.

Listing 6. HTML Document

Note the addition of the **?sortby=??????** parameters. We create new variables, make an initial assignment, and pass that, along with the document name (now called query2.html), to w3-msql. We use the **sortby** variable to contain a field name on which we wish to sort the listing.

Now that we have coded the skeleton for the front end, what changes are needed to the actual query template? Consider the rewrite of query.html, now appropriately called query2.html, shown in Listing 7.

Listing 7. HTML Document query2.html

The only major change is in the mSQL **select** statement. We added the standard ANSI SQL **order by** clause, passing along the content of our new **sortby** variable. In addition, note the use of the mSQL **print** command to display the sort field name in the header above the table. A **sortby** location displays the HTML table shown in Figure 3.

Figure 3. Concert Data Sorted by Location

To finish our simple query example let's revisit the concept of searching by location. Let's allow the user to input a city name manually, and pass it along to w3-msql for querying. Consider the simple HTML document, called ex3.html, shown in Listing 8. It provides an input field in a form, linked to w3-msql as the form processor.

Listing 8. HTML Document ex3.html

The query3.html document, which handles the actual query-by-city is shown in Listing 9. Note the change in the mSQL **select** statement. We use the contents of the form field in the ex3.html document as the value of a where-like SQL clause.

Listing 9. HTML Document query3.html

To test this form, enter **New York** as the city name—Figure 4 shows the output.

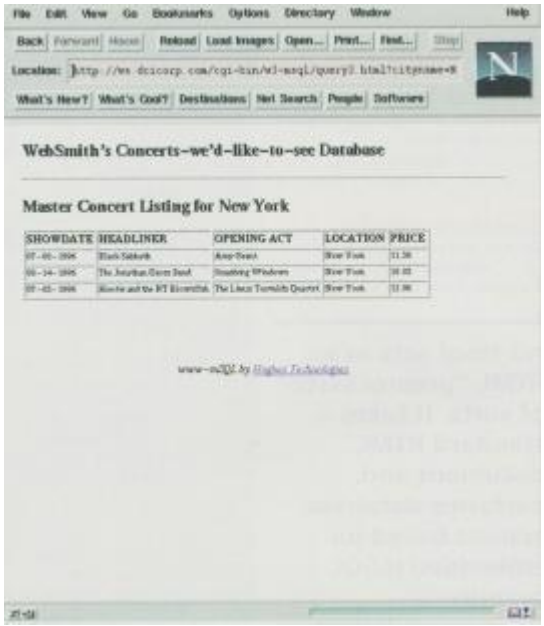


Figure 4. Form-based Query Results

That about sums it up for performing simple queries to a mSQL database. Try experimenting with different tables and interface designs. Try using frames. Place a search form in one frame and the query results in another. The possibilities are endless with the Linux/mSQL approach.

B. Scott Burkett formerly a Unix system programmer and technical instructor, is the Internet Services Manager for Decision Consultants, Inc, one of the largest software services consulting firms in the country (http://www.dcicorp.com). He enjoys major league baseball (Go Braves), good jazz bars, tinkering with Linux, and derailing military conspiracy plots in third world countries. He can be reached via e-mail at scottb@dcicorp.com.

LINUX JOURNAL

Creating a Multiple Choice Quiz System, Part 2

Reuven M. Lerner

Issue #38, June 1997

Designing our CGI quiz to be more robust and to include error checking.

Two months ago, we began to write a simple multiple-choice quiz engine in Perl. Virtually all of that column covered the nuts and bolts of the engine—creating the **QuizQuestions** object and the programs using that object to create a simple multiple-choice quiz, as well as to check its answers.

The end result was two CGI programs. The first, askquestion.pl, creates an instance of **QuizQuestions** and uses it to select a random question, which is then turned into an HTML form that is sent to the user's browser.

The other program, checkanswer.pl, accepts the submission of this form from the user, and then checks that the user chose the correct answer.

Even more important than the **QuizQuestions** object is the “quiz file”, an ASCII text file containing three different types of items:

- Comments beginning with a hash character (#). Comments are ignored by the quiz engine. Therefore, questions must not begin with **#**, but we can use **#** inside a question or answer without having to fear that the end of the question will be chopped off.
- Whitespace, such as spaces, tab characters and carriage returns that are also ignored. We allow for whitespace because users will undoubtedly separate items in the quiz file with blank lines, for example, and we need not require them to comment out the lines.
- Question records containing the questions and answers for the quiz. Each record contains the text of a question, followed by each of the four possible answers, and then by an A, B, C or D, indicating the correct answer. The fields in each record (question, answer 1, answer 2, answer 3,

answer 4 and the correct answer) are separated by tab characters, and so, neither questions nor answers can contain tabs.

A sample quiz file that tests users on their knowledge of the GNU Emacs text editor is shown in Listing 1. While this may not be obvious on paper, it is important to remember that the fields within each line are separated by tab characters, not by spaces.

One of the main flaws with the original quiz system was that it depended on the ability of users to create quiz files that conformed to these standards. Moreover, the **QuizQuestions** object didn't check for errors in format when reading the quiz file.

This month we take a look at how we can make the quiz system a bit more robust, while staying within the confines of the CGI standard.

Checking for Errors

First, we will modify the definition of **QuizQuestions** so that it checks for errors while loading the quiz file. What sorts of errors could we have it check for? One simple test ensures that each non-commented, non-whitespace line contains exactly six fields (one question, four answers and one answer key). Lines having a different number of fields will be flagged as errors.

Listing 2

The original version of **QuizQuestions.pm** is shown in Listing 2. To make sure that the quiz file is correct, we have to modify methods that read from the quiz file—which in this particular case, means the **new** method, the constructor for **QuizQuestions**. We can create a new instance of **QuizQuestions** with the following line:

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

Before we decide how to check for errors in the quiz file, we should think about how errors should be reported. If a method within **QuizQuestions.pm** discovers an error in the quiz file, should the method produce an HTML response for the user to see? Should it fail, calling **die** and indicating the error in the HTTP server's error log? Should it do both?

I suggest that **QuizQuestions.pm** should not use either of these options, since both violate the abstraction that we have created. **QuizQuestions** is an object for manipulating questions within a quiz file easily, and does not “know” whether it is being used from within a CGI program. Methods within

QuizQuestions should report errors, when they occur, to the calling program rather than directly to the user.

If we were using a language such as Java that includes an extensive exception-handling mechanism, this would be a perfect time to use it; we don't want the calling routine to receive a return value that could be misinterpreted as a legitimate value for **$quiz**. At the same time, we do want to return information about any errors that have occurred.

Perl's exception handling isn't as extensive as that of Java. Luckily, though, Perl does permit assigning various types of data to the same operator. In this case, if the file contains no errors, **new** returns a new instance of **QuizQuestions**. If there are errors in the file, **new** returns a string that consists of the line containing the error. It could simply return 0 in such cases; however, since we have the flexibility to return any scalar value, it is better to return a value that encodes more information.

Now that we have determined that error messages will be sent back to the calling method, let's think about how to determine which lines in the quiz file contain errors. Fortunately, this is a simple problem to solve, since each non-comment, non-blank line of a quiz file should contain exactly six tab-separated fields. Thus, if a line is not a comment, is not an empty line and does not contain six fields, it must be an error and should generate an error value.

Here is the loop in the existing version of **new** inside the **QuizQuestions** object that loads the quiz file from disk:

```
# Loop through the question file while (<QUESTIONS>)
{
   next if /^#/;      # Ignore comment lines
   next unless /\w/;  # Ignore whitespace lines
   chomp;
   # Add this question to the list.
   $questions[$counter++] = $_;
}
```

To check for errors, we simply break each line into its constituent fields using the **split** operator and count the number of list elements. If that number is not six, then we have a syntax error to be reported by returning the offending string to the calling routine. Here is a modified version of the above loop that implements this strategy:

```
# Loop through the question fil,e
while (<QUESTIONS>)
 {
    next if /^#/;      # Ignore comment lines
    next unless /\w/;  # Ignore whitespace lines
    chomp;
    # Split the line across tabs
    my @list = split(/  /);
    # Check to make sure that there are six fields
    if ($#list != 5)
```

```
    {
        # Return the line containing the error
        return $_;
    }
    else
    {
        # Add this question to the list
        $questions[$counter++] = $_;
    }
}
```

This code is the same as the original **while** loop with only one difference. Before adding the current line, **$_**, to **@questions** (an array containing questions and answers from the quiz file), we split it at each tab, creating a list with one element per field in the quiz file. If the list contains six elements, then this line of the quiz file is acceptable, and we continue with the original version of **new**--adding the current line to **@questions**, incrementing **$counter**, and moving on to the next line of the file.

If the list does *not* contain six fields, the line obviously contains an error. By the time we perform this test, we have already eliminated the possibility that the current line could be a comment or solely contain whitespace.

But wait a second—the caller is expecting to receive an object of type **QuizQuestions** in return. Because the **QuizQuestions** object can return many different kinds of scalar data, we have to make sure that the caller can determine whether the method invocation was a success (i.e., an object was returned) or a failure (i.e., a string was returned).

In this case, we use Perl's **ref** operator to find out if a scalar is a reference to an object and what kind of object it is. Invoking **ref** on a non-object scalar returns an empty string, which makes such testing easy. So, in the above version of **new**, we can create an instance of **QuizQuestions** with this code:

```
my $questions = new QuizQuestions("emacs");
&log_and_die($questions) unless (ref($questions)
         eq "
```

The second line checks to see if **$questions** is an instance of **QuizQuestions**. If not, we call **&log_and_die**, a routine (included in in Listing 5) that provides nicer logging of errors than a simple call to **die**.

While this code works, it makes for a poorly designed object. After all, why write the constructor so that the caller has to test the type of the object it returned? A better solution is to make **new** a minimalist creation method, and put the quizfile-loading mechanism into another method, called **loadFile**. This new method could then return either 0 indicating no error or a string containing the offending line.

With such methods in place, we write:

```
my $questions = new QuizQuestions("emacs");
my $error = $questions->loadFile;
&log_and_die($error) if $error;
```

This code creates an instance of **QuizQuestions** using the **new** operator, which does only the bare essentials. We load quiz file with the **loadFile** method. The **loadFile** method returns either 0, indicating that the file was loaded successfully, or a text string containing the line that caused a problem.

Since we modified **loadFile** to deal with errors, I have replaced the original uses of **die** which are inappropriate in a low-level object, (as mentioned earlier), with calls to **return**.

Rewritten versions of **new** and **loadFile** are shown in Listing 3.

Creating the Quiz File

So far, we have dealt with ways in which the **QuizQuestions** object can handle syntax errors within the quiz file. But many syntax errors are created simply by mistake or by users unfamiliar with the defined file format.

One solution is to provide users with tools for creating quiz files with fewer errors. Given the amount of time we spend writing CGI programs and HTML forms, it makes sense to create a short program that takes the contents of an HTML form and saves it to disk.

An example of one such form is shown in Listing 4. Upon submission, the form's contents are handed to create-quizfile.pl, which then creates a properly formatted quiz file.

In order to implement this feature, we need to add two new methods to **QuizQuestions**. One, **addQuestion**, takes a six-element list and adds it to **questions**, the instance variable containing fields from the quiz file. The second method, **saveFile**, does the opposite of **loadFile**, taking the current questions and saving them.

Here is one possible implementation of **addQuestion**:

```
sub addQuestion
{
    # Get ourselves
    my $self = shift;
    # Get our arguments
    my ($question, $a1, $a2, $a3, $a4,
                    $correct) = @_;
    # Turn our arguments into a string
    my $new_question = join("      ", @_);
    # Get our instance variable
    my @questions = @{$self->{"questions"}};
    # Add the new question
    push (@questions, $new_question);
    # Reset the instance variable
```

```
    $self->{"questions"} = \@questions;
    # Return successfully (= 0)
    return 0;
}
```

This version of **addQuestion** is fairly simple, if not very robust. For instance, it doesn't check to make sure the correct answer is one of A, B, C or D. But it does let us add new questions to the **QuizQuestions** object. Notice that **addQuestion** both retrieves and sets values for the instance variable **questions**.

If we were interested in extending our quiz on Emacs, we could use **addQuestion** in the following way:

```
my $error = $questions->loadFile;
&log_and_die($error) if $error;
$questions->addQuestion(
"What term describes the cursor's current location?",
  "mark", "point", "cursor", "mouse", "B");
```

Immediately after executing this code, **$questions** contains one more question. However, this question is lost upon the program's exit, because we have not yet saved the new question to the quiz file. In order to save the questions to a quiz file, define **saveFile** like this:

```
sub saveFile
{
    # Get ourselves
    my $self = shift;
    # Open the questions file for writing
    open (QUESTIONS, ">$questionDir" .
                     $self->{"quizname"}) ||
        return "Could not open " .
                     $self->{"quizname"} . " for writing";
    # Loop through the questions
    my @questions = @{$self->{"questions"}};
    my $question;
    for each $question (@questions)
    {
        print QUESTIONS $question, "\n";
    }
    close(QUESTIONS);
    return 0;
}
```

This code iterates through the questions, and writes them to the quiz file. Since we are writing all of the questions to disk rather than appending them, we use the **>** when opening the file, thereby overwriting any data that existed previously.

Since **saveFile** saves only the contents of the **questions** instance variable, it effectively obliterates comments and white space in the file. Of course, anyone creating the quiz file using a program is unlikely to look at the comments. Nonetheless, a more refined version of **saveFile** and the **QuizQuestions** object might let users add comments and white space to the file, as well as questions. (Obviously, the HTML form would also have to allow for this.)

Our version of **saveFile** uses the same system for reporting errors as **loadFile**--by returning a string, while the lack of an error is indicated by returning 0. This lets us use the following code:

```
$error = $questions->saveFile;
&log_and_die($error) if $error;
```

Now that you have seen the skeleton for create-quizfile.pl, you should have a good understanding of the program shown in Listing 5. This version of create-quizfile.pl is fairly straightforward. It checks to see if the user entered a question; if there is text for a question, it takes the remaining parameters from the HTML form submitted.

Now is a good time to remember that CGI programs that write user-defined strings to your file system are potentially dangerous, and thus must be placed in locations that are restricted to authorized users, either by using your HTTP server's built-in protection or by placing such programs behind a firewall. No matter how unlikely this may seem, a user may eventually discover that you have a program named create-quizfile.pl, and create quizzes on your system, possibly overwriting your creations.

This month, we made our quiz engine friendlier for non-programmers by checking the integrity of the quiz file and by allowing users to create quiz files using HTML forms. What happens when users want to edit quiz files? For now, they are stuck modifying the file on disk, which again opens Pandora's box of potential syntax problems. While we can discover these problems with our simple error-checking code, it might be a good idea to create a program that can edit quiz files as well as create them. Next month, we will modify create-quizfile.pl to do just that, making our quiz system easier for everyone to handle.

Reuven M. Lerner is an Internet and Web consultant living in Haifa, Israel, who has been using the Web since early 1993. In his spare time, he cooks, reads and volunteers with educational projects in his community. He can be reached via e-mail at reuven@netvision.net.il.

Archive Index Issue Table of Contents

Advanced search

Marketsmith

Doc Searls

Issue #38, June 1997

Flack Jacket: Observations by a renegade publicist from Doc Searls' on-line magazine, Reality 2.0.

Wired Wants to Fertilize Your Inner Potato

The March Issue of *Wired*, the computer industry's utopian fashion monthly, declares its wish to supplant the Web with media more suited to advertising. In a “special bulletin” by Kevin Kelly and Gary Wolf, and with its customary overstatement and retinal-torture colors, the magazine devotes its cover and the eleven pages that follow to “PUSH! Kiss your browser goodbye: The radical future of media beyond the Web”--a manifesto on the “emerging universe of networked media that are spreading across the telecosm.”

This universe doesn't exist yet, so *Wired* evokes its promise with the language of a sci-fi movie trailer: “Think video. Think text flickering over your walls. Think games that work. Think anything where a staid, link-based browser is useless.” Imagine “a zillion non-page items of information and entertainment.”

The Web, sadly, is just “an archive medium”. It's “a wonderful library,” *Wired* says, “but a library nonetheless.” Hey, who wants to surf in a public building? What today's Netizens really want is “a land of push-pull, active objects, virtual space and ambient broadcasting.”

This land won't rise from the work of the ordinary schlubs who now fill the Web's library with goods that offer rather than push. No, this Shangri-La will be summoned from the void by “20-year-old hot shots in the labs of PointCast, ESPNET, SportsZone and CNET.” Thanks to their pioneering genius, the browser is already “dying as the main event, to be reborn as a subsumed function and occasional option.”

Even though “the design of what is emerging...is now neither clear nor important,” *Wired* is sure it will let you “move seamlessly between media you steer (interactive) and media that steer you (passive).”

Let us pause to observe that the notion of media that steer you was never part of the Net's—or any other media's—appeal. The Internet's success owes exactly nothing to those who would use it for manipulation.

But I'll hand it to the editors of *Wired*: when they sell out, they go all the way. Not long ago, the magazine wrote an April Fool piece on “The death of the Web”. Now they mean it. “It turns out only a handful of us are up for the vigorous activity of reaching out to engage the world,” they say. “Most of us are still addicted to passive media.” Which, it turns out, is a good thing, because “there's a little couch potato in all of us.”

Pity the poor spuds forced to face “information framed on a two-dimensional hypertext page,” and to “navigate blind clickable links and search engine requests, drilling down to try to find what they want.” No wonder such labor-intensive tools are “retreating into the bowels of the Net.”

Wired wants your inner potato to enjoy “a more full-bodied experience that combines many of the traits of networks with those of broadcast.” This combination is what our young farmers call *push media*: “content is pushed to you, in contrast to the invitational pull you make when you click on the Web.”

“The central mission” of this new agriculture “is to shoot every conceivable media flavor across, through, in between and around a network that includes every possible hardware device.” What you get will be “in-your-face, immersive, experiential push media.”

The editors say “at first glance all this looks a lot like the revenge of TV.” At last, from beneath the Web's riot of voluntary vegetation “the subterranean instincts of couch potatoes rise again!”

Wired tells us “push” has been with us since the Dawn of Art. In journalism, for example, “once you dive into a story...the author pushes you along, and the magazine steers.” At movies you “surrender to the director's manipulation of your emotion and your mind.” Indeed, art is a pushy business.

And thankless too. Consider the pitiless consumers whose constant rejection the pushers endure. “In the competitive jungle of 500,000 channels”, today's pushiest media are subject to “the relentless interactive tick of the zapper.”

“That almost neurotic urge to zap,” *Wired* says, “has falsely led people to think that what viewers want is more zapping, more control, more steering. What they want instead are more ways to zap.” Enter “the emerging postbrowser interfaces” that “create different ways to *play* human attention.”

Sadly, the Web is too demanding for the average spud. “Web users suffer a sense of being lost and overwhelmed.” More than half of them have given up surfing, *Wired* laments, because they just hit the same old sites, or they find “the signal is camouflaged by all the noise.”

That noise is made by spuds whose web content is second-rate, or worse, kind of a hobby. They've turned the Web into a craft fair of bad watercolors and lopsided ceramic bowls, where such in-your-face ties as those created by the artisans at *Wired* are all too rare.

“Yeah, rolling your own is very rewarding, but often we'd like someone else to slip us a ready-made. Even though it may not be as nifty as the one we made. Or maybe because it is niftier and better made.”

Trust your inner potato. “*Seinfeld* viewers know what we're talking about.” Television succeeds for a good reason. “You dial it for a mainline fix of unadulterated push. It's great for that universal plunge into the Thing Larger Than Myself.” (*Seinfeld*? Are we that small?)

And so “we are now about to arrive at television (push media), before we finally emerge into what interactivity is really about.”

Ah, so all this push business is really just television, only better—more “ambient”, more “ubiquitous”, more “sexy”, more of a “warm, familiar, fuzzy convenience”. Well, that brings us to the big question: Who is going to pay for all this?

Advertisers, of course. “Advertisers and content sellers are very willing to underwrite this,” *Wired* says, even though they would be wrong to “happily back push media in hopes that the spells that work on TV will work there too.” But hey, advertisers are easily hypnotized, too. “Push has advertisers transfixed,” the editors say. “In the short run, advertisers can be counted on to pile in.”

This from a company that hasn't been able to sell an IPO because even the ravenously credulous investors who are driving stock prices to the sky don't buy their magazine's inflated estimates of itself.

Well, let me tell you about advertising.

Before some of today's push geniuses were born, I co-founded what is now one of Silicon Valley's biggest advertising agencies. My name is still on the door. That agency does many millions of dollars in annual billings, mostly in print. Some of it, I suppose, runs in *Wired*.

Advertising is a product of scarce access to large numbers of customers and prospects. Since the Dawn of Advertising, The Media have been the sole providers of that access, and they've charged a lot for it.

But when companies find ways to interact directly with customers and prospects, they will shoot resellers, distributors, retailers, advertising media and every other margin-demanding intermediary that stands in the way. In fact, the shooting has already started. The result is a new trend called *disintermediation*. It's a lot more scary than downsizing, because it starves whole companies and business categories, rather than just a few employees.

The most threatened businesses are the ones that depend on advertising. This is why the last thing ad-supported media want is an efficient market for product information. But that's exactly what the Net provides.

The lights *Wired* sees at the end of the Web's dark tunnel are miner's lamps of countless companies digging toward their customers. They're digging with the same Internet tools *Wired* now demeans, including the Web, browsers and e-mail. These tools cost squat, and they do an amazing job.

In the face of this, advertising has an existential choice: help dig or get shoveled aside.

The shoveling will be easy. The advertising we all know and hate is a huge and often wasteful expense that most CEOs can downsize or ax with few immediate penalties. Some executives and suppliers might lose their jobs, but the SEC and IRS won't even notice.

And don't think the Big Boys aren't looking carefully at the issue, even if they share *Wired*'s da-glo wet dreams about “immersive media” and “ambient advertising”.

Two years ago at PC Forum, John Warnock of Adobe observed that ads in *The Wall Street Journal* waste lots of trees and deliver no obvious results, while a single notice on the company's web site brings thousands of downloads and countless useful customer relationships. And Warnock is one of the guys who appreciates image building, branding, positioning and other advertising arts.

The Net shortens the distance between supply and demand, both for products and for information. What *Wired* calls “a vast unmapped cave of documents” is the most powerful marketing resource ever created, because it can deliver deep and rich marketing content on demand—especially once it's mapped.

Does *Wired* really think “spelunking” with search engines is the terminal stage of network organization, and that what traditional LANs call “directory services” won't show up soon? (For clues about where all this going, check out Netscape's *LDAP White Paper* and *a Bulldozer Through the Intersection* in *Reality 2.0*: http://www.batnet.com/searls/bulldozer.html.)

There is something huge happening on the pull side of more markets than Thomas Register can list. It's not creating dumb web pages and not surfing around for entertaining distractions. It's demanding useful information, contacting suppliers directly, and treating as garbage everything that doesn't add value. That garbage will include much of today's advertising and the media that depend on it—unless, of course, they evolve to life forms that survive what in TechnoLatin we might call “the emerging demand-driven information environment”.

By demeaning both the supply and the demand sides of markets for everything other than narcotic entertainment, *Wired* sets new altitude records for stupidity and arrogance. If the magazine had any moral content in the first place—and I have to believe they did, given a masthead that still includes Stewart Brand, John Perry Barlow, Esther Dyson, John Heileman and other advocates of freedom and self-worth—they forfeited it with this contemptible tract.

What *Wired* editors hail is really a postmodern *Invasion of the Body Snatchers*. It treats Netizens not only as vegetables, but as hosts for its advertisers' pods.

Fortunately, advertisers live in reality. I hope they give these fools a dose of it.

Doc Searls is President of The Searls Group, a Silicon Valley consultancy, and a co-founder of Hodskins Simone and Searls. He has been writing on technology and other issues for most of his life. The Flack Jacket series of essays are collected in *Reality 2.0*, http://www.batnet.com/searls/docworks.html). Other

series are *Positions* and *Milleniana*. He can be reached via e-mail at searls@batnet.com.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #38, June 1997

Readers sound off.

Where is the E-mail Address?

Just got my comp copies today of the 3/97 issue. Thanks very much. It's always a thrill getting those first copies of a published work.

I am very disappointed on one count, though. My e-mail address was removed from the short bio at the end (page 72). All my esteemed fellow authors included their e-mail addresses, so I assume it was an unconscious mistake on some editor's part. (Not to excuse it; careless mistakes are the most preventable and least forgivable variety in my book.)

Funny thing is, I'm decidedly unimpressed with authors who avoid interaction with their readers. They abdicate the stewardship of knowledge their work otherwise earned them. Now I'm involuntarily guilty of this very sin. The main reason I write articles is to try to make contact with forms of intelligent and enlightened life out there. I missed my chance this time. I guess my lesson for the next time is to insist on a review copy.

In all other respects, it's been a real pleasure working with you folks. Mary Webber, by the way, is wonderful. —Bob Stein bobstein@earthlink.net

Polygon Code

Much to my surprise, I found that in the article “A Point About Polygons” by Bob Stein in the March 1997 issue of *Linux Journal*, the code in Listing 2 (TESTPOLY.C) was specifically written for Turbo C and the DOS environment. As it was apparently coded back in 1995, one may assume that Stein has since discovered a more mature development platform, but I do hope this is not to be taken as a subtle shift of focus on the part of the editors of *LJ*. —Robert V. Schipper rvs@gol.com

Author Responds

Thanks for writing, Robert.

You're quite right, TESTPOLY.C was for the Borland/DOS environment. At Galacticomm we've been using Borland's command line development environment for some time, and we continue to do so for both the DOS and NT versions of Worldgroup. So at the time, it was convenient, taking me only an hour or two to use.

If you're hoping I've seen the light and started using Linux I'm afraid I'll disappoint you. Lately I've been using Microsoft's Visual C++ and Sun's Java for graphics programming. I assume my article was accepted for its Websmithing theme [Yes, that's why—Ed.]. —Bob Stein bobstein@earthlink.net

No Dialtone on Modem

I was reading the article titled “Setting Up UUCP” in issue 35, and noticed the author was wondering why his modem dials when there is no dial tone. I have discovered over the years that some modems return:

```
NO DIALTONE
```

while others return:

```
NO DIAL TONE
```

So, including a line:

```
chat-fail NO\sDIAL\sTONE
```

would probably solve his problem. —Scott Barker scott@mostlylinux.ab.ca

MostlyLinux

I am very disappointed that after twice sending information to you guys about my company (the second time was actually about a dozen copies of the same e-mail, sent every week or so until I finally got a reply), you still got it wrong. My company is MostlyLinux, but my entry in the *Linux Journal 1997 Buyer's Guide* lists me as “Calgary UNIX Users Group”, with my phone numbers, but with their snail-mail address instead of mine, and my e-mail address through them rather than through my own company.

This is going to cause confusion both for myself and the Group (with whom I volunteer, and on whose behalf I have dealt with SSC, which may have caused confusion on your part). I remain a loyal reader of *Linux Journal* (which I find

very useful), but am very unhappy that I am going to have to deal with the problems this creates. For future reference, if you intend to publish another *Buyer's Guide*, please note that I am: —Scott Barker MostlyLinux, Inc. Voice Mail: 403-209-9406Fax: 403-285-1399E-mail: info@mostlylinux.ab.ca URL: http://www.mostlylinux.ab.ca

Craftwork Solutions

We just received our copy of the *Linux Journal 1997 Buyer's Guide*. I feel this type of effort is very good for the industry. Craftwork Solutions is focused on making Linux an accepted commercial solution for businesses. We are glad to see SSC make the effort to explain to the general public the benefits of using Linux.

What did trouble me came at the end of the issue. Craftwork Solutions announced back in Sept/Oct '96 our 2.2 release for both the Intel and Alpha architectures. We were across the aisle from SSC at Comdex in November '96, showing our 2.2 releases. Unfortunately, your table included only the out-of-date information on our 2.0 product.

You made room for both the 3.0 and 4.0 releases of Red Hat. I would have expected that at least our 2.2 information would have been used! Craftwork Solutions has advertised with *LJ* since 1995. I would very much like to understand how this oversight occurred.

Your publishing of our old data makes us look like a company that is not concerned about the direction of the industry and not interested in providing the best product and support to its customers. I personally take that very hard. My staff worked long weeks during the summer to have the new releases ready for Comdex.

I realize the information we filled out for you back in May '96, reflected the 2.0 product. What confuses me is that the Red Hat 4.0 wasn't available back in May '96 either. Please explain to me how this mixup occurred, and how we can prevent it from occurring in the future. —Lee Morse, Chief Technology Officer lmorse@craftwork.com Craftwork Solutions, Inc.

Data is Our Life

This was our first buyer's guide and we made some mistakes, but we learned from them and plan to have an even better issue next time. One of the things we are most concerned about is data gathering methods. For this issue, other than the sunsite listings, we printed only what was sent to us. If you did not send in updated information, we would not have updated it for you. Red Hat obviously did send in updated information. If you did send in updated

information, then I apologize for the table not getting updated. Actually, in either case, I apologize. Next time, we'll include a check for the latest distributions in our procedure. We do know what the current distributions are.

File Locking Services

Mr. Kraft's comments in the March 1997 issue of *Linux Journal*, regarding Linux's lack of network file locking services, are dead on the mark; however, I would now like to make it publicly known that there is an ongoing development effort to provide a lockd and statd for Linux.

This effort is currently combined with an effort, led by Olaf Kirch, to revise major portions of the Linux NFS implementation. A kernel-space lockd, written by Olaf, and a user-space statd, initially written by me and then significantly modified by Olaf, are currently part of Olaf's NFS development distribution “snapshots”.

A developers' mailing list exists for people who wish to contribute to, or participate in the alpha/beta testing of, this development effort. The list address is lockd-statd@linux.nrao.edu, and the list's subscription address is majordomo@linux.nrao.edu. Current snapshots of the linux-nfs development code can be retrieved from the following anonymous FTP directories:

ftp.mathematik.th-darmstadt.de:/pub/linux/okir/dontuse/ linux.nrao.edu:/pub/people/linux/okir/dontuse/

There are currently plans to publish an introduction to network file locking, together with a description of the Linux implementation, in an upcoming issue of *Linux Journal*. In addition, I will be giving a short presentation on this subject at the April 1997 Linux Expo in Raleigh, North Carolina. —Jeff Uphoff juphoff@nrao.edu

Security Issues

If you are going to do a security article, get it right. People get cgi and suid programs wrong on their own without your printing an article that contains serious errors. A good article on cgi security would have been just what is needed. Unfortunately this wasn't it.

Let's take this:

```
exit(system("/home/foo/www/bin/counter.sh"));
```

If I run this handy provided example by doing:

```
cd HACKDIR
cp /bin/hash ./home
ln -s suidxi—program ./foo
IFS='/'
export IFS
 ./foo
```

I get a shell as the person it is **setuid** to.

Why? Because the system runs the command through the shell, and the shell uses IFS as its “white space” definition.

This is basic setuid security stuff.

The procmail-based example at least does use a magic cookie to stop fake mails. It has other bugs; notably, it forgets sendmail may deliver multiple mails in parallel using data, but then I guess it makes it plain it's just trying to show the trick, not do it right. —Alan Cox alan@cymru.net

On-Line Linux Users Group

Hi. I have been a longtime reader of *LJ* and it has been a great help to me, and I am sure that applies to many in the Linux Community! Now, my friends on the Net and I have also done something as a contribution to Linux which I thought would be interesting to you and helpful to your readers. We have created an On-Line Linux Users Group for people interested in learning more about Linux, providing help to other Linuxers, and promoting Linux:

http://www.linuxware.com/ —Peter Lazecky peter@linuxware.com

Archive Index Issue Table of Contents

Advanced search

LG and IELG

Marjorie Richardson

Issue #38, June 1997

I recently did an e-mail interview, in the guise of Editor of Linux Gazette, for the Italian Edition of Linux Gazette.

I recently did an e-mail interview, in the guise of Editor of *Linux Gazette*, for the Italian Edition of *Linux Gazette*. I know it sounds strange, but the Italian edition is basically our *LG* with a few additions such as this interview. (I really wasn't interviewing myself.) Since I wanted to talk about *LG* this month anyway, I thought presenting the interview would be a good way to do it. The questions were presented to me by Francesco De Carlo, a member of the faculty of Computer Science at University of BARI, Italy and the Director of the Italian Edition of *Linux Gazette*, which can be found at www.media.it/LUGBari/index.html. Mr. De Carlo can be reached via e-mail at fdecarlo@mailbox.media.it.

Mr. De Carlo: When and why did SSC decide to publish *Linux Gazette* in the current version? Originally, *LG* was edited only as an extra-curricular activity by John M. Fisk.

Margie: During the summer of 1996, John Fisk decided he no longer had the time to keep *Linux Gazette* up in the fashion it deserved. *LG* had become very popular, and readers were wanting it to come out on regular monthly basis. Between school and work, John just didn't have time to do this, so he put out feelers looking for someone to take it over. We responded and he accepted us as the right people to continue *LG*.

SSC responded to John because we had always felt that *Linux Gazette* was a worthy and necessary asset to the Linux community. We did not want to see it either go away or be taken over by someone who would turn it into a commercial enterprise. We promised John that *LG* would remain free and it has.

Mr. De Carlo: What kind of relationship does the *LG* have with his “big brother” *Linux Journal*? Some exchanges of articles, writers, ...?

Margie: Yes, *Linux Gazette* and *Linux Journal* do a lot of sharing. As of February 1 of this year, I am Editor of both *Linux Journal* and *Linux Gazette*. Every month we use an article from *LG* in *Linux Journal*, and occasionally, I will use articles from *LJ* in *LG*--usually those about conferences and other events surrounding Linux. And yes, I have authors who write for both magazines, most notably the regular contributors of columns to *LG*: Larry Ayers, John Fisk and Michael Hammel. *Linux Gazette*'s Answer Guy, Jim Dennis, has done an interview with Stronghold's Sameer Parekh, which will be appearing in the July issue of *Linux Journal*.

Mr. De Carlo:: Are authors wishing to write for *LG* contacted by you or do they send articles to you? That is: do you prepare a list of the subjects that will be discussed in the next issue of *LG*, or can users send you any article, on any topic?

Margie: *LG* is managed very casually; authors can send me articles on any topic and I will include them. Whatever comes in during the month goes in the next issue. There is no focus other than Linux. Also, I do not edit the articles; they are posted just as the authors send them.

Mr. De Carlo: Are you alone in producing *LG*? Or do you have a real “editorial office” with real “editors” and “reporters”? If yes, how do you make it function?

Margie: I have no real editors or reporters to help. I depend on outside authors in the Linux community to make their contributions, and the wonderful thing is, they do. Some months I have more material than others (January was really packed), but I've never been short. I have gotten a lot of help with graphics and HTML from SSC's webmaster, Michael Montoure. Beginning this month, I have a new assistant, Amy Kukuk, who will be helping out by doing the *News Byte* column and perhaps more.

Mr. De Carlo: What are your plans for the near future? Introducing a new *LG* with a renewed graphic look, new articles and so on?

Margie: I intend to continue posting *Linux Gazette* each month and promoting it wherever I can. I feel it is even more of an asset than ever to both new and experienced Linux users.

Our look seems to change periodically. With the March issue, we dropped the spiral that caused so many problems. Michael is inventive, and we mainly add things as we come up with them.

We have two new columns that will be appearing regularly, “The Answer Guy” by Jim Dennis, and “Clueless at the Prompt, A Column for New Users”, by Mike List. Both columns are good for new users looking for help.

Linux Gazette is free for the readers, but is not free for SSC. To help defray the publishing cost, *LG* has begun accepting sponsors. A small acknowledgment of these sponsors will be made on the Front Page. Our first sponsor is InfoMagic—our thanks to them for their help.

Mr. De Carlo: What do you think about our LGEI? Is it a good idea and, above all, can it help Italian Linux users to better understand this OS?

Margie: I think LGEI is wonderful! It'a great way to spread the word about Linux to all Italy. With our regular columns and articles, as well as all the tips and tricks people send us, I feel LGEI is an invaluable resource to Italian Linux users, just as our English version is to Linux users worldwide.

Marjorie Richardson, Editor

Archive Index Issue Table of Contents

Advanced search

UniForum '97, April 12-14, 1997

Marjorie Richardson

Issue #38, June 1997

My trip to San Francisco to attend UniForum'97 was very satisfying as I got to see two great luminaries of our time—the Hale-Bopp comet and Linus Torvalds.

My trip to San Francisco to attend UniForum'97 was very satisfying as I got to see two great luminaries of our time—the Hale-Bopp comet and Linus Torvalds. Hale-Bopp was visible in the pre-dawn sky on March 12 and 13. Linus was visible at the Keynote speech on March 13 and was definitely the brighter of the two.

The president of UniForum, Tom Mace, was present to welcome Linus, and Douglas Michaels of SCO presented Linus with UniForum's Achievement Award. The award itself is a clear, pyramid-shaped trophy, about which Linus said he was pleased to have something “physical” to show for his work. Linus' acceptance speech was brief and self-effacing as usual. He referred to himself as the “spider at the center of the web” with many others working around him. Tove and their 3 month old baby girl, Patricia Miranda, had accompanied Linus and both tolerated my pushiness in taking pictures. After the keynote, Linus and Tove made the rounds of the Exhibit Hall, visiting all their fans in the Linux Pavilion. Tove confided that they were enjoying the weather (no snow), but that the arrival of their furniture had been delayed by a bad storm that had forced the ship back to Germany.

Mitchell Kertzman of Sybase gave a vibrant keynote speech that morning, in which he ignored Linux as a possible factor in a paradigm shift that might topple Microsoft. Perhaps he hasn't heard that Linus' goal is “world domination”. Kertzman compared today's software industry to the automobile industry of the fifties—that it is designing products to be obsolete in three years, while consumers are wanting long term reliability. Sounds to me like consumers are looking for Linux.

While 7000 people had pre-registered for UniForum, only about 75% of those actually attended. Perhaps the others went to one of the competing shows such as Internet World. At any rate, at times the floor was crowded with attendees, while at other times (particularly toward the end of the day) it was quite empty. The Linux Pavilion was placed in the right rear corner of the floor, yet it seemed to me that most attendees were gravitating over to check out this upstart operating system that dares to be freely available. SSC gave away their stock of magazines and bumper stickers, as well as displaying T-shirts, reference cards and the new “Tux” mugs. IBM and Lucent Technologies both had central positions on the floor, but I saw many people passing them by to visit Digital to check out both the Alpha and Jon “maddog” Hall's new Linux setup for Digital's Intel box. Jon is providing us with a short article about this setup that will appear next month.

I attended two of the talks: one on Electronic Document Interchange and one on high speed Internet access. Both were well presented and full of good information. I was particularly impressed with Jeff Wilbur's thoughts on the directions that access to the Internet will take in the future (i.e., cable modems, xDSL, satellite, ISDN), and so asked him for an article.

Since UniForum '97 was my first conference as Editor of *Linux Journal*, I met many people I had only heard about before, including Joel Goldberg of InfoMagic (who is a sponsor of *Linux Gazette*), Mark Bolzern of WGS, Adam Richter of Yggdrasil, and of course, Jon “maddog” Hall of Digital. Jon introduced me to Ted Cook of BRU, who told me of his plan to give away BRU software to Linux User Groups at the upcoming Linux Expo and to groups that are members of G.L.U.E. (http://www.ssc.com/glue/).

On Wednesday night Joanne Wagner, one of our advertising representatives, and I attended a press conference/party put on by XiGraphics—free food and drink, always a plus. The press conference was held to announce the recent name change (from X Inside) and the latest release of Xi's Accelerated X software. The president and founder of the company, Thomas Roell, gave a short presentation in which he described the directions he envisions for Xi Graphics.

All in all, I had a good time at the conference and a pleasant stay in San Francisco.

Marjorie Richardson, Editor

Archive Index Issue Table of Contents

Advanced search

Traveling Linux: An Implementation Experience for Unattended Management Applications

Maurizio Cachia

Issue #38, June 1997

Linux runs the trains in northeast Italy.

SAD operates in the local public transport field in an alpine area in the Northeastern part of Italy. We spent several years constructing an integrated payment system for all public transport operators (buses and trains) in our region. We now manage this payment system using about 90 PCs running 2.0.x Linux. As part of a new automatic vehicle location project, we plan to set up special industrial PC systems with GPS (Global Positioning System) and radio for data transfer on-board of every bus. On this Linux system, we implement procedures for traffic and tariff control.

An Integrated Payment System

About two years ago the Provincia Autonoma di Bolzano (The Autonomous Province of Bolzano—see sidebar) started to use a standardized payment

system that allows patrons to use one ticket for all methods of public transportation, either on road or rail. This encouraged the setting up of an integrated information system with the aim of standardized management of all information obtained from statistical data about passengers and trips.

The system is based on the use of on-board electronic devices, with magnetic technology, which were built by a Belgian company with international experience in this field, and it required more than four years to completely set up.

The service network is managed by 25 different companies with very different needs and organizational models. In order to have standardized system management we developed a single software model that can define its own operational features, adapting to different conditions, and can integrate the management procedures for the bookkeeping and operational control of the principal companies.

The limits of the hardware features and the operational and development system were taken for granted from the first steps of implementation. These limits were apparent when it became necessary to merge the lines managed by the Ferrovie dello Stato (National Railroad Company) into our system.

This integration required us to set up devices like those on board the vehicles in 37 rail stations. These systems run unattended; therefore, it is necessary to have tele-diagnostic functions, remote updating and automatic data collection available.

Linux on Rail

About the middle of 1993, our consultants told us about the existence of a Unix-like operating system with source available on the Internet that had a license policy similar to that of the Free Software Foundation.

Our introduction to Linux, at that time on the 0.98 release, excited us at once because of its great stability and because the source was available if the need arose to make modifications in order to develop our own projects. Thus, we could implement an agreement with our current system house to obtain the updated and tested Linux distributions and to develop the device drivers for our not exactly standard devices. We modified the devices responsible for running of the magnetic records, converting them back to their natural terminal function, by which they are linkable to a personal computer through a strong and reliable transmission file.

In the first months of 1993, we set up the devices inside the stations. After a testing period and following the availability of the first 1.0.x Linux release versions, the whole system was put into action.

Today, 90 systems are running overall:

- 40 wholly unattended inside the rail stations,
- 40 by the ticket windows in the most important stations of the country, and
- 10 inside the bus depots, that read and collect data coming from the buses through infrared transmissions.

Every unattended system consists of a black box set in a burglar-proof case that contains a 486 33MHz PC with 16MB of RAM and a 150MB hard disk, a minimum of four serial ports (UART 16550A), a V32bis or V34 modem and a battery backup. To each black box are connected (in RS 485) from two to six ticket obliteration units like those on the buses. These units now use Linux release 2.0.2x (ELF version).

The average system uptime is now more than 100 days; some system crashes still take place, probably because of particular environmental conditions. To obviate this, we are testing a hardware watchdog in some places. The ticketing systems are, of course, augmented by the presence of a monitor, a keyboard and terminal connections needed for credit card treatment.

Every night the central system connects to all the terminals, via a Taylor UUCP, for data collection and to start out some functions. Every week we use a PPP connection to do the systems clock adjustment using **ntpdate**.

Linux on the Bus

The precise knowledge of the patronage mobility features is not enough to solve every management problem of a bus company. In this industry employees are spread throughout the land and can be reached for service instructions only at particular times and at precise network points. Thus, programming the employee and bus schedule ahead of time is important, but does not allow for time lost in the work organization due to network problems, accidents, organizational difficulties or unexpected demand changes.

Therefore, we did a survey on the checking methods and technologies used in other countries. In general, these systems have been applied to the urban services of mid-sized metropolitan areas. The check service ends at the border of the suburbs, mostly because of the high investment costs needed for the

radio transmission equipment. Architectures are normally based on these general features:

1. Almost all check intelligence to a central point that is able to process a great quantity of data (in general, computers with real-time, special operation systems);
2. Availability of a radio network with a great number of transmission channels and widespread cover, using a polling transmission method with every vehicle questioned to determine its position at very short time intervals (in general, less than a minute);
3. In case of limited radio cover, integration with infrared active captor systems or with passive markers to reach vehicles or to allow them to correct their position;
4. The intelligence on-board is collected by devices with industrial electronics features and proprietary technology based on a moderate capacity CPU.

In recent years, the check systems described above have made an evolutionary jump following the availability of a location system, based on the GPS technology, in the civil market. This system, although limited in very crowded areas, simplifies the vehicle location electronic systems with a noticeable increase in accuracy.

Based on the above experiences, the system project features were defined for the intercity and urban services in the Provincia di Bolzano, which operate under the following conditions:

- The control network covers a length of 2300 km, of which more than three quarters is in an alpine area;
- The regulation and structural situation existing in the national spread of radio frequencies does not allow a sufficient number of channels to construct a polling system for the land in question;
- The existence of several different companies, with different organizational structures, means there must be control models with very simple features, but always consistent with a standard system management; therefore, the overall system functionality can be linked to a single control center that can define all relevant vehicles and send out operation instructions;
- The existence of the magnetic payment system at its current development level requires its integration inside a single on-board system that will allow more developments and implementations as needed.

In this case, it seemed necessary to set up on every vehicle a knot of the operation system that could develop, in strict autonomy, the production management functions, while maintaining the overall integrity of the system

and requiring the least amount of resource spending. Functions had to be modular—that is, put into action according to the needs of the company that owns the vehicle and provides the service. Every vehicle must be able to determine its own position, time and program consistency, without requiring a constant link between vehicle and operation center.

Each working program change that could be foreseen for a vehicle and its operator had to be described with a minimum of verbal communication, using the same standards as the instructions of the central information system of the company. The only solution that fulfills these requirements is a multitasking operational environment that can be integrated with the company information system (based on DG/UX 5.4.3, the implementation of Data General CO of Sys V rel.4). The purchase of an Intel-based commercial system for more than 400 vehicles would require license investments of more than a hundred million lira.

Moreover, the need for non-standard features in the trade systems (particularly terminal use, watchdog management and advanced power management) might make it necessary to intervene at a system level, with all the possible troubles between the software distributor (Italian) and the owner (American).

In this framework, Linux is the only operating system that works for our project. The features we considered in Linux's favor are as follows:

1. System steadiness. With the 1.2.13 (and better in the stable 2.0) the machine average uptime is much more than that for any of the other widely circulating Unix systems for the Intel platform that we tested. (And we tested almost all of them.)
2. Source code availability and quality. Many of our special needs are already contained in the development version (i.e., power management, watchdog, networking on radio net, etc.). The others will be met by either modifying the source ourselves or through cooperation with Linux developers.
3. Our management applications were born in a Unix environment. From 1980 until today, we had to face porting both to different architectures

and to different systems (Ultrix, BSD, SCO, Interactive, DG/UX). The porting to Linux was the easiest and the most linear.

4. Linux is free. This was the most basic reason that helped us to persuade our management that Linux was the system to use.

We finished the setting up phase of the automatic vehicle location system in July, 1996. Following a European call for bids, Data General (Westboro, MA) was selected as supplier for the on-board PCs. The technical specification for the data transmission system had to be integrated into the already existing radio network.

Each vehicle's equipment is composed of a 486/100MHz PC, with 16MB of RAM, a 540MB hard disk, 10 RS232/422 serial ports, a SCSI controller and a type III PCMCIIA. The same PC case (140 mm x 140 mm x 158 mm) also contains a GPS Trimble differential system and an intelligent management unit that allows programmed system ignition, environmental functional parameter control and battery backup functions.

The systems were tested for very hard environmental conditions, since they must function in temperature between -20 to +50 Celsius degrees, and be shock resistant following military specifications (MIL-SPEC).

The PCs were linked with the current on-board terminals used for payment system management (obliterators and issue console), the communication equipment (radio or GSM, Global System for Mobile communications) and special public information panels.

Our current operating system is Linux 2.0.25.

Maurizio Cachia lives in a little village in the Dolomiti Alps with his wife and a funny golden retriever named Lu. He has worked since 1980 as a system analyst in the Unix environment for the public transport companies. In 1984 he became the Technical Manager of the Integrated Information System of SAD in Bolzano. He can be reached by e-mail at mau@sad.com.

Archive Index Issue Table of Contents

Advanced search

Booting the Kernel

Alessandro Rubini

Issue #38, June 1997

This article is a description of the steps required to boot the Linux kernel. While this kind of information is not relevant to the system's functionality, it is interesting to see how the different architectures bring up the system.

A computer system is a complex machine, and the operating system is an elaborate tool that orchestrates hardware complexities to show a simple and standardized environment to the end user. When the power is turned on, however, the system software must boot the kernel and work in a limited operating environment. I describe here the booting process of three platforms: the old-fashioned PC and the more fully featured Alpha and SPARC platforms. The PC is covered in more detail, since it is still in more widespread use than other platforms, and also because it's the most tricky platform to bring up. No code will be shown, as assembly language is unintelligible to most readers, and each platform has its own.

The Computer at Power-On

In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is “unmovable software” found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or “flash” to stress its hardware implementation, while others call it “console” to focus on user interaction.

The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system. I don't deal with firmware issues here with the kernel code, which is distributed with Linux.

The PC

When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as “real mode” and is dictated by compatibility with older processors of the same family. Everything that makes up a complete system must live within the available megabyte of address space, i.e., the firmware, video buffers, space for expansion boards and a little RAM (the infamous 640KB) must all be there.

To make things difficult, the PC firmware loads only half a kilobyte of code and establishes its own memory layout before loading this first sector. Whatever the boot media, the first sector of the boot partition is loaded into memory at the address 0x7c00, where execution begins. What happens at 0x7c00 depends on the boot loader being used; we examine three situations here: no boot-loader, LILO, Loadlin.

Booting zImage and bzImage

Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command **cat zImage > /dev/fd0** works perfectly on Linux, although some other Unix systems can do the task reliably only by using the **dd** command. Without going into detail, the raw floppy image created by **zImage** can then be configured using the **rdev** program.

The file called **zImage** is the compressed kernel image that resides in **arch/i386/boot** after either **make zImage** or **make boot** is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a “big **zImage**” instead, the kernel file created is called **bzImage** and resides in the same directory.

Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a **zImage** kernel in detail; all of the following path names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.

- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called “zero-page”, used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run.

The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer the Linux kernel, instead the “gunzip” part of the **gzip** program resides at that address. The following additional steps are now needed to uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of “gunzipping” the kernel; it calls the function *decompress_kernel*, defined in compressed/misc.c, which in turns calls *inflate* which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the “real” mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed boots) can complete processor initialization and call **start_kernel()**. Code for all functions after this step is written in C.

The various data movements performed at system boot are depicted in Figure 1.

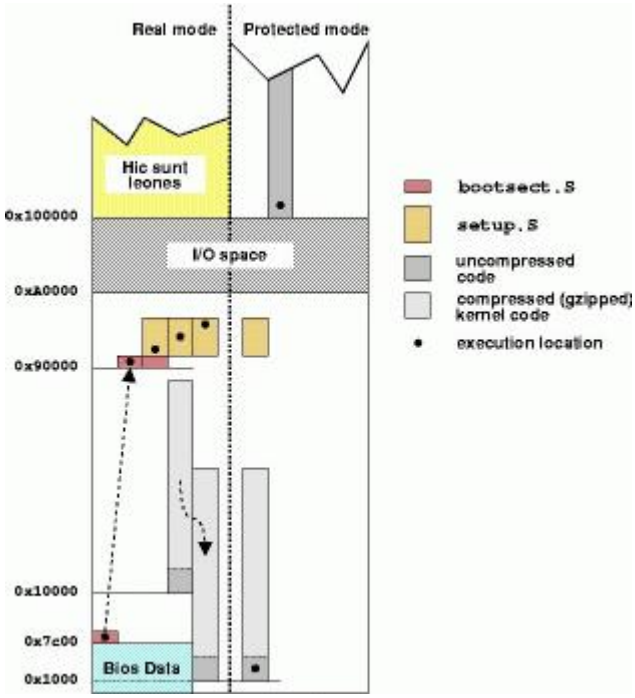


Figure 1. System Boot Data Map

The boot steps shown above rely on the assumption that the compressed kernel can fit in half a megabyte of space. While this is true most of the time, a system stuffed with device drivers might not fit into this space. For example, kernels used in installation disks can easily outgrow the available space. Some new method is needed to solve the problem—this new method is called **bzImage** and was introduced in kernel version 1.3.73.

A **bzImage** is generated by issuing **make bzImage** from the top level Linux source directory. This kind of kernel image boots similarly to **zImage**, with a few changes:

- When the system is loaded to address 0x10000, a little helper routine is called after loading each 64K data block. The helper routine moves the data block to high memory by using a special BIOS call. Only the newer BIOS versions implement this functionality, and so, **make boot** still builds the conventional **zImage**, though this may change in the near future.
- setup.S doesn't move the system back to 0x1000 (4K) but, after entering protected mode, jumps instead directly to address 0x100000 (1MB) where data has been moved by the BIOS in the previous step.
- The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the address 0x100000 (1MB). Several memory moves are needed to perform the task correctly.

The rule for building the big compressed image can be read from Makefile; it affects several files in arch/i386/boot. One good point of **bzImage** is that when kernel/head.S is called, it doesn't notice the extra work, and everything goes forward as usual.

Using LILO

Most Linux-x86 users don't boot the raw kernel image from a floppy; instead they boot LILO from the hard disk. LILO replaces part of the process outlined above so that it can load a Linux kernel that is scattered throughout a disk. This capability allows the user to boot a kernel file from a file system partition without using the floppy.

In practice, LILO uses the BIOS services to load single sectors from the disk, and then it jumps to setup.S. In other words, it arranges the memory layout in the same way as bootsect.S; thus, the usual booting mechanism can complete painlessly. LILO is also able to handle a kernel command line, and this is a good reason by itself to avoid booting the raw kernel image.

If you want to boot a **bzImage** with LILO, you must use LILO version 18 or later. Earlier versions of LILO are not able to load segments into high memory, an ability that is needed when loading big images in order for **setup.S** to find the expected memory layout.

The main disadvantage of LILO is that is uses the BIOS to load the system. This forces the kernel and other relevant files into the first 1024 cylinders of disks to be accessible to the BIOS. When using the PC firmware, you discover how old-fashioned the architecture actually is.

Even if you don't run LILO, you can enjoy the documentation files distributed with LILO's source code. They document the boot process on the PC and explain how to handle (almost) every conceivable situation.

Using Loadlin

If you want to boot your operating system from another operating system, Loadlin is the tool for you. This program is similar to LILO in that it loads the kernel from a disk partition and then jumps to setup.S. It is different from LILO in that it not only faces the BIOS restrictions, but also must dispose of an established memory layout without compromising the system's stability. On the other hand, Loadlin is not restricted to a half kilobyte length because it is a complete program file, not a boot sector. Version 1.6 and later of Loadlin are able to load big images.

Loadlin can pass a command line to the kernel and is, therefore, as flexible as LILO. Most of the time, you'll write a linux.bat file to pass a full-featured command line to Loadlin when calling the **linux** command.

Loadlin can be used to turn any networked PC into a Linux box. All that is needed is a kernel image equipped for mounting the root partition via NFS, Loadlin and a linux.bat containing the correct IP numbers. You need a properly configured NFS server as well, but any Linux machine can fill that job. For example, the following command line turns a PC (alfred.unipv.it) into a workstation:

```
loadlin c:\zimage rw nfsroot=/usr/root/alfred \
nfsaddrs=193.204.35.117:193.204.35.110:193.204.35.254:255.255.255.0:alfred.unipv.it
```

More of It

The code is not as easy as I described—it must deal with a lot of details, such as bringing around the kernel's command line, keeping an eye on the boot technique being used, and so on. The curious reader can look in the source file to learn more and to read the authors' comments. There's a lot of information in the comments, and they are often funny to read.

I personally feel most users will never need to touch the boot code, because things are much more interesting when the system is up and running. At those times you can exploit all the features of your processor and all the available RAM without going mad with processor-level issues.

Booting an Alpha

The Alpha platform is much more mature than the PC, and its firmware reflects this maturity. My experience with Alpha is limited to the ARC firmware, which is the most widely used.

After performing the usual detection of devices, the firmware displays a boot menu that lets you choose which file to boot. The firmware can read a disk partition (though only a FAT partition), so you actually boot a “file” without the need to hack boot sectors and build maps of disk blocks.

The file booted is usually linload.exe, which in turn loads MILO (the “Mini Loader”). In order to boot Linux through the ARC firmware, you must have a small FAT partition on your hard drive to store linload.exe and milo files. The Linux kernel doesn't need to access the partition unless you upgrade MILO, so FAT support can be safely left out of your Alpha kernel.

Actually, the user can exploit different options. The ARC boot menu can be configured to boot Linux by default, and MILO can be burnt in flash memory in

order to get rid of the FAT partition. However, whatever you do, you end up with MILO running.

The MILO program is a stripped-down version of the Linux kernel. It has all of the Linux device drivers and a file system decoder; unlike the kernel it doesn't have process control and does include Alpha initialization code. This tool can set up and enable virtual memory and can load a file from either an *ext2* partition or an *iso9660* device. The "file" in question is loaded to virtual address 0xfffffc0000300000 and then executed. This virtual address is also the one where the Linux kernel runs; however, it's unlikely you'll ever load anything but Linux. One exception is the **fmu** ("flash management utility") program used to burn MILO in flash ROM—**fmu** is compiled to execute from the same virtual address whence the kernel runs, and it is distributed with MILO.

It's interesting to note that MILO also includes a small 386 emulator and some of the PC BIOS functionality. This is needed in order to execute self-initialization code found on many ISA/PCI peripheral boards (PCI boards, though claiming to be processor-independent, use Intel machine code in their ROM images).

Since MILO does all of this, what is left to the Linux kernel?—very little, actually. The first kernel code to execute in Linux-Alpha is arch/alpha/kernel/head.S, and all it does is set up a few pointers and jump to start_kernel(). Actually, kernel/head.S for Alpha is much shorter than the equivalent x86 source file.

If, for some reason, you don't wish to run MILO there is an alternative, though not a practical one. In arch/alpha/boot you'll find the source for a "raw" loader that is compiled by issuing **make rawboot** from the top level Linux source directory. This utility can load a file from a sequential region of a device—either floppy or hard disk—using the firmware's "call backs".

In practice, the raw loader accomplishes a task similar to the one bootsect.S performs for the PC platform—it forces a copy of the kernel to either a raw floppy or a raw hard disk partition. There's no real reason to use this technique—it is quite hairy and lacks the flexibility MILO offers. I personally don't know if it still works; the "PALcode" used by Linux is exported by MILO and is different from the one exported by the ARC firmware. The PALcode is a library of low-level functions used by Alpha processors to implement low-level hardware management like paging; if the current PALcode implements different operations than the software expects, the system won't work.

Booting a SPARC Station

Bringing up a SPARC computer is similar to booting the Alpha on the user side, and similar to booting the PC on the software side.

The user sees that the firmware loads and executes a program, which in turn is able to retrieve and uncompress a file found on a disk partition. The “program” in question is called SILO, and it can read files from either a *ext2* or a *ufs* partition. Unlike MILO (like LILO), SILO is able to boot another operating system. There is no need for this ability on the Alpha, because the firmware can boot multiple systems; once you run MILO, you have already made your choice (the right choice—Linux).

When a SPARC computer boots, the firmware loads a boot sector after performing all the hardware checks and device initialization. It's interesting to note that Sbus devices *are* platform independent, and their initialization code is portable Forth code rather than machine language bound to a particular processor.

The boot sector loaded is what you find in /boot/first.b in your Linux-SPARC system and is a bare 512 bytes. It is loaded to address 0x4000, and its role is retrieving /boot/second.b from disk and writing it to address 0x280000 (2.5 MB); this address was chosen because the SPARC specifications state that at least 3MB of RAM must be mapped at boot time.

The second-stage boot loader then does everything else. It is linked with libext2.a to access system partitions and can thus load a kernel image from your Linux file system. It can also uncompress the image, since it includes the inflate.c routine from the **gzip** program.

The routine second.b accesses a configuration file called /etc/silo.conf, similar in shape to lilo.conf. Since the file is read at boot time, there's no need to re-install the kernel maps when a new kernel is added to the boot choices. When SILO shows its prompt, you can choose any kernel image (or other operating system) specified in the silo.conf file, or you can specify a complete device/path name pair to load a different kernel image without editing the configuration file.

SILO loads the disk file to address 0x4000. This means the kernel must be smaller than 2.5MB; if it is larger, SILO will refuse to overwrite its own image. No conceivable Linux-SPARC kernel currently exceeds that size, unless it was compiled with **-g** to have debugging information available. In this case, the kernel image must be stripped before being handed to SILO.

Finally, SILO performs kernel decompression and/or remapping to place the image at virtual address 0xf0004000. The code that takes over after SILO is finished is arch/sparc/kernel/head.S. The source includes all the trap tables for the processor and the actual code to set the machine up and call **start_kernel()**. The SPARC version of head.S is quite big.

start_kernel and On

After architecture-specific initialization is complete, the init/main.c program takes control of whichever processor you are using.

The **start_kernel()** function first calls **setup_arch()**, which is the last architecture-specific function. Unlike other code, however, **setup_arch()** can exploit all the processor's features and is a much easier source file to deal with than those described earlier. This function is defined in the kernel/setup.c code under each architecture source tree.

The **start_kernel()** function then initializes all the kernel's subsystems—IPC, networking, buffer cache and so on. After all initialization is done, these two lines complete the code:

```
kernel_thread(init, NULL, 0);
cpu_idle(NULL);
```

The **init** thread is process number 1: it mounts the root partition, executes **/linuxrc** if **CONFIG_INITRD** has been selected at compile time, and then executes the **init** program. If **init** can't be found, **/etc/rc** is executed. In general, using **rc** is discouraged, since **init** is much more flexible than a shell script in handling system configuration. As a matter of fact, version 2.1.21 of the kernel removed the **/etc/rc{/}** option, making it obsolete. If neither **init** nor **/etc/rc** will run or if they exit, **/bin/sh** is executed repeatedly (but 2.1.21 and later kernels will execute it only once). This feature only exists as a safeguard in case the **init** file is removed or corrupted by mistake. If you remove a.out support from the kernel without recompiling your old **init**, you'll enjoy having at least a shell running after reboot. The kernel has no more tasks to do after spawning process number 1, and all other functions are handled in user space by **init**, **/etc/rc** or **/bin/sh**. And process 0? The so called “idle” task executes **cpu_idle()**, a function that calls **idle()** in an endless loop. **idle()** in turn is an architecture-dependent function that is usually in charge of turning off the processor to save power and increase the processor's lifetime.

> Alessandro is a Linux enthusiast who writes documentation because he's not smart enough to write software. His 486 is specialized in grepping through source code, and humbly leaves real jobs to the Alpha and the SPARC. He can be reached via e-mail at rubini@ipvvis.unipv.it.

Archive Index Issue Table of Contents

Advanced search

New Products

Amy Kukuk

Issue #38, June 1997

Integra4 RDBMS for Linux, Clustor 1.0, BitWizard and more.

Integra4 RDBMS for Linux

Integra4, a relational database management system, has been announced by Coromandel Software Ltd. Integra4 is compliant with ANSI SQL2. Some of its features include: engine supporting referential integrity, stored procedures and triggers, forms, 4GL application development and embedded SQL support. There are also other tools developed around Integra4 which include Bankon, RA, GIS tools and a portfolio management system. There is a special pricing policy for the Linux Community.

Contact: Coromandel Software Ltd., N-505, North Block Read Wing, Manipal Centre, Dickenson Road, Bangalore 560042, INDIA Phone: +91(80) 5585463, Fax: +91(80) 5586415, E-mail: cosoft.developer@gems.vsno.net.in.

Clustor 1.0

Active Tools, Inc. announced the release of Clustor 1.0, a program for managing large computational tasks. Clustor simplifies a common computationally intensive activity—running the same program code numerous times with different inputs. Clustor provides increased performance by distributing jobs over a network of computers and improved task management through a friendly user interface. Clustor 1.0 is currently available for several operating systems, including Linux. Introductory prices range from $799-$1299 for Clustor Root and $199-$299 for Clustor Node. Discounts for multiple copies are available. Evaluation versions of Clustor can be downloaded from: http://www.activetools.com/.

Contact: Active Tools Inc., 246 First St., Suite 310, San Francisco, CA 94105 Phone: (415) 882- 7062, Fax: (415) 680-2369, E-mail: info@activetools.com, URL: http://www.activetools.com/.

BitWizard

BitWizard is pleased to announce that it is starting a Linux device driver service. This means you can concentrate on creating PC-based systems, and BitWizard will make the required device drivers for the cards you select. The driver will be ready within a week or two after BitWizard gets the hardware and the documentation. For price quotes contact BitWizard.

Contact: BitWizard, Phone: +31-70-3700841 or +31-654-727520, E-mail: R.E.Wolff@BitWizard.nl, URL: http.//BitWizard.nl/.

Hitachi MP-EG1A

Hitachi will soon have MPEG Cameras available for evaluation and review. This is a point-and-shoot camera. The MPEG Cam features include: MPEG movies, shoot and store 3000 JPEG images, and digital audio. The system requires a multimedia PC with a Pentium 100Mhz or faster, 16MB RAM, 8MB available on hard disk, and a CD-ROM with speakers or headphones. The cost of the MPEG Camera is $2,499.95.

Contact: MPEG Cam Network Webmaster, Phone: (305)443-7973, Fax: (305) 447-0745, E-mail: Rob@QuantumLeap.net, URL: http://www.MPEGCam.net/.

LinkSca

Electronic Software Publishing Corporation announced LinkScan, the first commercially available link checker that operates on Unix servers. Designed to work on both Internet and Intranet servers, LinkScan can test over 30,000 links per hour because it uses multi-threaded simultaneous processing. For a web site, LinkScan offers the following features: Scans for missing HTML documents, images and other files; validates all internal hyperlinks; checks all name tags and references, creates two types of site maps or table of contents suitable for publication and discovers orphaned files. The price for personal use is $49.95 each. A license is required for each server on which the product is used.

Contact: Electronic Software Publishing Corporation, 1504 #8-00200 Main Street, Gardnerville, NV 89410-5273, E-mail: linkscan@elsop.com, URL: http:// www.elsop.com/.

Pixel!FX Version 5.1

Mentalix Inc. announced the availability of Pixel!FX Version 5.1. Additionally, the entire Pixel!FX Version 5.1 suite of products is now available for Linux users. Mentalix has also released a new version of its Pixel!SCAN Photoshop Plug-In, which provides scanning functionality for users of Adobe Photoshop for Unix.

Pixel!OCR 5.1 builds on the capabilities of previous versions with its multiple OCR engine technology. Certain modules may be purchased separately, or users may choose the entire, end-to-end imaging suite in one product, Pixel!FX Deluxe.

Contact: Mentalix 1700 Alma Drive, Suite 110, Plano, Texas 75075, Phone: 972-423-9377, Fax: 972-423-1145, E-mail: info@mentalix.com, URL: http://www.mentalix.com/.

JClass Chart

KL Group Inc. announced the launch of JClass Chart. JClass Chart is a Java Bean component that enables Java developers to embed sophisticated graphs and charts into applications and applets quickly and easily. JClass Chart joins JClass BWT and JClass LiveTable Pro in KL's Java component offerings. Some features include: rotation and scaling capabilities, flat files and sockets, and axes andlabels for easy development. The cost of the JClass Chart is $399 and the Chart Source is $999.

Contact: KL Group Inc. 260 King Street East, Toronto, Ontario Canada M5A 1K3, Phone: 800-663-4723, Fax: 416-594-1919, E-mail: info@klg.com, URL: http://www.klg.com.jclass/chart/.

Internet/Intranet Design Shop

Boomerang Software announced the Internet/Intranet Design Shop. The program integrates a WYSIWYG HTML document processor, graphics editor, photo editor, business graphics and presentation creator, a clip art manager and a calendar generator. A free trial version is available for downloading from the company's web site at http://www.mosaiccom.com/.

Contact: Boomerang Software, Phone: 617-489-3000, E-mail: info@mosaiccom.com, URL: http://www.boomerangsoftware.com/.

Velocis Database Server

Raima Corporation announced it is now offering Velocis Database Server. Velocis is a client/server database management system for the Linux operating system. Velocis supports multiple database models. Velocis offers SQL C-API so developers and end users can use an industry standard interface for manipulating and controlling the database. It also enables developers to partition their applications using Server Extensions, which allow application code to be hosted directly against the database server.

Contact: Raima Corporation, 4800 Columbia Center, 701 Fifth Ave., Seattle WA 98104, Phone: 800-327-2462, 206-515-9477, Fax: 206-748-5200,URL: http://www.raima.com/.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #38, June 1997

Our experts answer your technical questions.

Deleting User Accounts

I want to delete a user account. But I can't find any command to do so. Is there any utility to do so? —Harry Wong

A Simple Process

Many of the Linux distributions ship with a **deluser** or **userdel** command that reverses the action of the **adduser** or **useradd** command. Search the man pages or simply try the commands to see if they exist on your system.

Failing that, deleting a user consists of two main steps:

1) Delete the user's entry in **/etc/passwd**

As root, and using your favorite editor, edit **/etc/passwd** (I always make a backup copy before messing with the password file because you can never be too careful). Search for the line that starts with the users login ID and delete the entire line.

2) Delete the user's home directory.

Again as root, use the rm command recursively to get rid of files by typing — Vince Waldon vwaldon@redcross.ca

Using MD

I have just added four additional drives to my Linux/Compaq Proliant 1000. I recompiled the kernel to include the md/linear option. Now I cannot find any instructions on how to make the MD work with all the drives. What I want to do

is have all five drives connected such that when the first drive is full, the data will go to the second drive and so on. —Robert Binz

Getting Help with MD

More information on using MD can be found in the MD FAQ at ftp://sweet-smoke.ufr-info-p7.ibp.fr/pub/linux/. You should find two files. The first, **md-FAQ**, can answer many of your questions. The second, **md035.tar.gz** (this is the current version of MD at the time of this writing), contains the utilities you will need to manage your MD system, as well as more documentation.

Be aware the MD package is still under development. Certain parts of the system (such as mirroring) are not yet considered stable. If you plan to use MD, I recommend you join the mailing list by sending mail to majordomo@vger.rutgers.edu with the message body "subscribe linux-raid". —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Restricting Users to FTP Access

I need to create a captive account which restricts the user to only FTP access of the system. I also need to restrict the user to accessing only directories above a root directory I specify. Can you please let me know how to implement this? —Steve Stuczynski

Creating Guest Groups

First add a guest group entry in the **/etc/ftpaccess** file. Specify which group of users will be treated as guests by typing **guestgroup ftponly**. Then create the ftponly entry in **/etc/group** by typing **ftponly::22:**. Next, create a user member of this group, with no shell, in **/etc/passwd**:

```
user1:the_passwd:22:22:Limited FTP user:/home/ftp/user1:/bin/true
```

*Don't forget to create the **/home/ftp/user1** directory. Last, add **/bin/true** in **/etc/shells**. Now check your work to make sure it works! —Pierre Ficheaux, Lectra Systemes pierre@rd.lectra.fr*

Secure FTP

There are simple and complex ways to restrict user access to FTP only. There is a HOWTO that describes this in detail, as well as potential security problems you should be aware of.

This FAQ is unfortunately not an official part of the Linux HOWTO and mini-HOWTO compilation, but Slackware users can find it as part of the installation anyway, in /usr/doc/faq/howto/mini/Anon-FTP-FAQ. Although the document is

geared primarily towards creating a secure anonymous FTP site, it actually covers an extensive range of the setup required for your desired effect. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Changing Configuration Settings

After a successful installation of Red Hat's Colgate release of Linux, I have found I would like to change some of my configuration settings. Is there a way to get back into the setup utility that steps you through setup? Or is there an easier way of doing this? In particular, my NIC is not working right and I don't know how to configure it correctly. —Jeff L. LaPlante

Using Control Panel

You need to use the control panel. It is an X-based set of utilities. The control panel will start automatically if you do a **startx** as root, or you can do a **su**, set your DISPLAY environment variable, and then run **control-panel**. In particular you want to run the Network Configurator (**netcfg**) and possibly the Kernel Configurator —Donnie Barnes, Red Hat Software redhat@redhat.com

Changing g++ Filenames

How can I change the name of the output file after compiling my source code with g++? I don't have the manual entry for this command. —Kennie Jose Cruz

Renaming Executables With -o

To change the name of an executable created by g++ or gcc, use the following command: —Rafael Rodrigues Obelhei roobelix@mikrus.com.br

Info Files and Man Pages for gcc

You can find this information in the gcc info files which should be accessible by typing **info gcc**, or in a shorter version by typing —Ralf Stephan

Keeping Track of Version Changes

Where can I find out what changed between Linux kernel versions? —Koen Rosseau

The Kernel Change Summary

Check out the Kernel Change Summary at ftp://ftp.shout.net/pub/users/mec/kcs/. This covers the 1.3, 2.0, and 2.1 series kernels. —Matt Hartley hartlw@rpi.edu

Xterm Error Message

When trying to run xterm under X, I get the error message **no ptys available**. I have used Slackware in the past and have never had a problem with xterm before. —Thomas Granger

Restoring Device Files

Most likely some of your pty device files got messed up. Check in **/dev** and restore them with **mknod** or —Bert Vermeulen bert@terra.cnct.com

Finding bootp

Where do I find a bootp server software and directions on how to install it? —Carl Fritch

Check Man Pages

Any Linux distribution should come with a bootpd (probably either /usr/sbin/bootpd or /usr/sbin/in.bootpd) and a man page for it. —Steven Pritchard, Southern Illinois Linux Users Group steve@silug.org

Archive Index Issue Table of Contents

Advanced search