Advanced search

Linux Journal Issue #105/January 2003

Get Your Home Network on IPv6 Today!
LINUX JOURNAL
39 Hot New Zaurus Apps
ENGAGE!
IPv6 WEB SITES with Apache
SWITCHING to JFS
COMPILING JAVA with GCJ
VIDEO CONFERENCING with GnomeMeeting
FreeS/WAN: Fast VPNs on a budget
Star Trek Nemesis Goes Linux

Archive Index

Advanced search

Get IPv6 Now with Freenet6

Peter Todd

Issue #105, January 2003

How to set up an IPv6-over-IPv4 tunnel and provide IPv6 networking to hosts behind a gateway.

IPv6 is the successor to our current internet protocol, IPv4. It offers many new features, including a vastly increased address space (128 bits of address vs. IPv4's measly 32 bits), easier autoconfiguration and better support for encryption. The Debian Project has done a good job of making its distribution IPv6-ready. Here, I show you how to set up an IPv6-over-IPv4 tunnel and provide IPv6 networking to hosts behind a gateway using the free Freenet6 tunnel service and radvd. Note that I'm assuming you're using Debian Woody (3.0) or unstable (SID) and that you understand how to install packages through apt. I'm also assuming you know a little bit about IPv6, such as what an address looks like.

You'll need a publicly addressable IPv4 address on the gateway you're using. The kernel on that gateway and any hosts you want to connect also must support IPv6; Debian's standard kernels support IPv6 by default. If you're not sure whether yours does, check for a /proc/sys/net/ipv6 directory. Also, see if the output of **ifconfig** contains any IPv6 addresses, such as fe80::24f:49ff:fe07:2552. Make sure your firewall isn't blocking IPv6 tunnel packets; IP protocol 41 (ipv6 over ipv4) must not be blocked in either direction.

IPv6 tunneling will fail if you're behind a NAT router, but the good news is that you don't need a static IPv4 address from your service provider to have a static IPv6 address.

First of all, install the Freenet6 client software with apt. The Debian package is called freenet6 and is available in Woody and later releases. RPM and source packages are also available from www.freenet6.net/download.shtml. Once it's installed, Freenet6 automatically gets an IPv6 address based on your existing IPv4 address; no configuration is required.

The /etc/init.d/freenet6 script simply starts the tunneling client, called Tunnel Setup Protocol Client (tspc). At this point, if your IPv4 address changes, so does your IPv6 address. That's right, simply by installing Freenet6, everything will work right out of the box on most systems. However, we want to go a step further; we want to get our own IPv6 subnet. To do this, we need a user account.

Go to www.freenet6.net, and click on the “Create your account” link, which should be under TSP Server on the left-hand menu. After filling in your user name and e-mail address, you will get mail containing the password for the user ID you requested.

Open /etc/freenet6/tspc.conf in your favorite editor. This configuration file controls tspc and defaults to making an anonymous connection. Find the user ID and password lines and change their values to the ones in the e-mail. If you restart Freenet6, run **/etc/init.d/freenet6 restart** and stop here; you'll have a perfectly functional IPv6 setup with a single static IPv6 address. If this is all you need, you can stop reading right now.

Assuming you do need more, the next step is to request your /48 prefix. Again, this is done by editing tspc.conf. Add the following lines to the end of your tspc.conf file:

```
host_type=router
prefixlen=48
if_prefix=eth0
```

The if_prefix option controls which interface will be regarded as your internal network. Freenet6 automatically enables IPv6 forwarding between this interface and your Freenet6 tunnel. On most setups, eth0 will be correct. However, you can change this as required. For the rest of this article your internal network interface will be referred to as eth0. If your setup uses another interface, use that interface instead.

At this point, you can restart Freenet6; **run /etc/init.d/freenet6 restart** to load the new configuration. We're not going to be making any more changes to it. If you run **ifconfig** after restarting, you'll notice a new IPv6 address on eth0 in the form XXXX:XXXX:XXXX:1::1/64. Freenet6 automatically has given eth0 this new address from your /48 prefix. The XXXX:XXXX:XXXX part is the network part of your /48 prefix. You may want to make this address permanent by adding it to the eth0 section in /etc/network/interfaces. This will allow other hosts to find your gateway even if the Freenet6 software isn't running. To do this, find the section for eth0, which should look something like this:

```
iface eth0 inet static
        address 10.1.1.1
        netmask 255.255.255.0
        network 10.1.1.0
        broadcast 10.1.1.255
```

Add these lines (replacing 3ffe:0b80:083b:1::1 with the address Freenet6 assigned, without the /64 netmask) below:

```
iface eth0 inet6 static
        address 3ffe:0b80:083b:1::1
        netmask 64
```

Test this with **ifdown eth0** and **ifup eth0**. One trick: if you're doing this remotely and eth0 is the interface you're connecting through, you can avoid losing your connection and having to walk to the actual computer you're working on by chaining those two commands together with a semicolon, **ifdown eth0 ; ifup eth0**. Your intrepid author did exactly this over an SSH connection (my test box is downstairs; I'm upstairs).

If everything's working fine up to this point, we can start on the radvd setup. IPv6 has a lot more support for autoconfiguration than IPv4. However, for this autoconfiguraton to work, the routers of a network must answer the autoconfiguration requests of the hosts. On Linux, the program to do this is called radvd (Router ADVertisement Dæmon). The actual autoconfiguration works by giving hosts new IPv6 addresses based on the 64-bit network address and their hardware Ethernet address. Hosts also are told what their default gateway should be. As with Freenet6, there is a Debian package for radvd.

Once radvd is installed, we need to edit the /etc/radvd.conf file. Debian's default configuration basically does nothing, so replace the whole lot with:

```
interface eth0
{
   AdvSendAdvert on;
   prefix 3ffe:b80:840:1::/64
   {
      AdvOnLink on;
      AdvAutonomous on;
   };
};
```

The prefix should be set to the network and netmask on eth0. For instance, if the IPv6 address Freenet6 assigned to eth0 is 3ffe:b80:83b:1::1/64, the prefix should be set to 3ffe:b80:83b:1::/64, which is the same as the IPv6 address only minus the host part of the address (the :1). Once you've done that, restart radvd.

After restarting, try running **ifconfig** on IPv6 hosts attached to your internal network. You should notice a new IPv6 address on the interface connected to your internal network. For example, using the prefix above, one such address might be 3ffe:b80:840:1:24f:49ff:fe07:2552/64. If no new address appears, check /var/log/syslog on the system on which radvd is running for errors; radvd does not log to the console.

To test your IPv6 connection, first try pinging something with the ping6 program on both the gateway and a connected host. You might find you don't have ping6 installed. If not, install the iputils-ping package. Some hosts to try ping6-ing include www.6bone.net and www.kame.net. If this isn't working, double-check that your firewall isn't blocking IPv6 packets. If this works, try connecting to www.kame.net with lynx, Mozilla or Konqueror. These browsers have IPv6 support in Debian testing. If everything is working, there will be a dancing “kame” at the top of the page and a little message at the bottom. If it still says you're using IPv4, and the ping6 test worked fine, double-check that you have the latest version of your web browser.

Installing an IPv6 tunnel may open up security holes. While IPv6 firewalling using iptables in 2.4.x kernels is possible, it is beyond the scope of this article. You are *strongly* advised to evaluate possible security issues. One quick way to check whether any IPv6 services are being opened up is to run the command **netstat -l -A inet6**. This will show you open, listening IPv6 sockets. If any are active, make sure it's okay for them to be publicly available.

Congratulations! You're on the next-generation Internet. Now that you have a basic IPv6, you will probably want to look into IPv6-enabling services such as Web and e-mail. The Debian IPv6 Project is a good start to find patched versions of Debian packages. While many client programs are already IPv6-compatible, there are a lot of basic server dæmons, such as inetd, apache, etc., that aren't in the mainstream Debian release. These packages are easy to install; add a few lines to your APT configuration, and run **dist-upgrade**. If you're interested, go to people.debian.org/~csmall/ipv6.

Resources

Peter Todd has been using Linux since he was 14. When he's not messing around with upcoming network protocols, he can be found at Woburn Collegiate working on pottery and film arts projects. He can be contacted at pete@petertodd.ca and has a self-hosted IPv4 and IPv6 web site, petertodd.ca.

Archive Index Issue Table of Contents

Advanced search

Linux and Star Trek

Robin Rowe

Issue #105, January 2003

Robin speaks with the studio Digital Domain on using Linux to render special effects in Star Trek Nemesis and other films.

Linux got its first big Hollywood break in 1997 when Venice, California studio Digital Domain (D2) used Linux to render the special effects for the hit movie *Titanic*. We spoke with D2 while they were in production using Linux with *Star Trek Nemesis*, which has a scheduled release date of December 13, 2002. D2 uses Linux for both renderfarm servers and artist desktops.

D2 has used Linux for 21 motion pictures, including best visual effects Academy Award winners *Titanic* and *What Dreams May Come*. D2 has won two Scientific and Technical Achievement Academy Awards: one for Track motion tracking software and the other for the compositing software NUKE.

Like most studios, D2 was primarily using SGI hardware running SGI's IRIX variant of UNIX, both on renderfarm servers and artist workstations. Experiments at D2 with *Dante's Peak* in 1996 proved that a move to Linux was feasible. “The Linux renderfarm came first”, notes D2 Digital Production and Technology Creative Director Judith Crow. “With *Titanic* we were working with a company called Areté using Renderworld, their ocean-simulation software. It ran three times faster on our Linux Alphas than on our IRIX SGI machines.” While the renderfarm paved the way, applications such as NUKE and Houdini pushed Linux to the desktop.

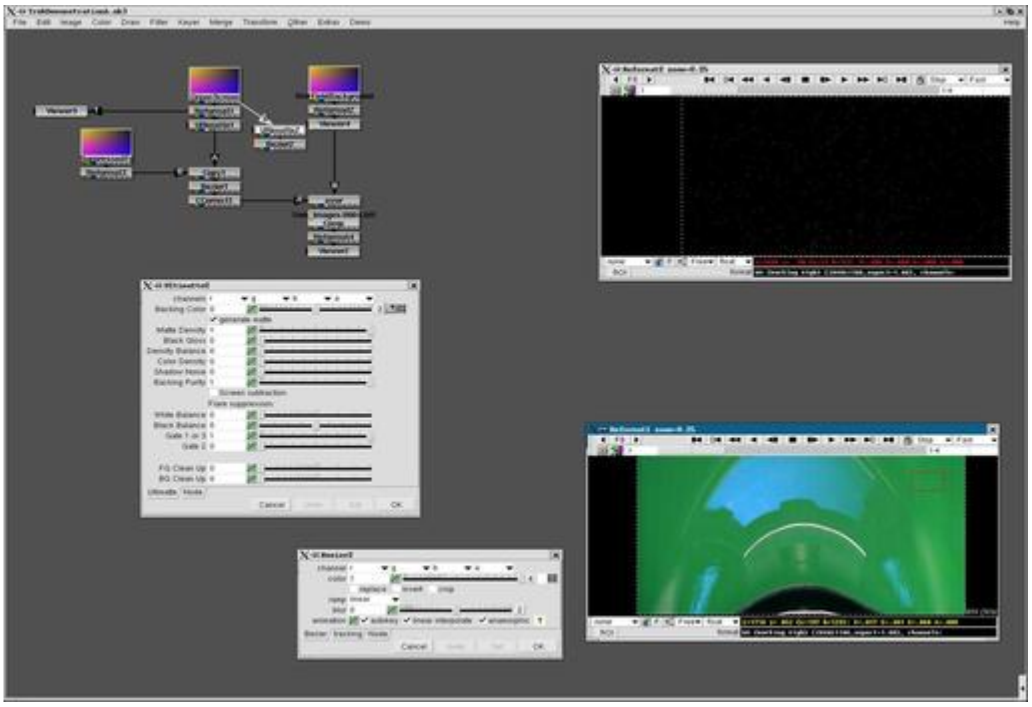


Figure 1. Preparing to insert a star field into the window of a spacecraft using NUKE. The tree graph to the right is a nodal view of the composite script.

A compositor is what software artists use for overlaying moving images, for example, the starship *Enterprise* flying past a background matte of a space station. “Digital Domain has been running NUKE on Linux since 1997 when it was used extensively on *Titanic*”, says Digital Effects Supervisor Jonathan Egstad. Egstad, along with D2's Bill Spitzak, Paul Van Camp and Price Pethel received an Academy Award for the NUKE compositor.

“NUKE is essentially a 2-D renderer”, says Egstad. “It is five or six times faster on Linux than IRIX, but it wasn't until the beginning of 2001 that the Linux GUI was able to run fast. Back in 1993, NUKE was the original scanline-based design. It only took 20MB of RAM to render a typical composite instead hundreds of megabytes.” Later commercial compositor applications, such as Shake, the popular node-based compositor sold by Apple, have a similar design.

“There are many instances where 2-D can assist in the workload”, points out Egstad:

> We can build a complete 3-D scene in NUKE then refer to that in a 3-D package like Maya and vice versa. A 3-D scene can be created and rendered in Nuke3, complete with lighting, texturing and shader support—diffuse, Blinn and Phong are built-in. There's a complete 3-D subsystem in NUKE. That's a trend in all 2-D packages. 2-D packages are more and more turning into 3-D packages.

Houdini, a commercial 3-D package of which D2 is a big user, offers its own integrated compositor called Halo in its latest version. As with NUKE, it is hierarchy-based in conjunction with 2-D hierarchy. D2 also uses the commercial 3-D packages LightWave and Maya.

FLTK, the Window Toolkit of NUKE

NUKE version 3 has been in use at D2 since 2001, running on Linux, IRIX and Windows. D2's first Linux renderfarm was on Digital Alphas and still gets some use. The NUKE design retained the keystrokes used in IRIX, so users, especially freelancers working at D2, wouldn't face a learning curve when moving between operating systems. “The NUKE interface is deliberately Spartan, designed more toward feature work”, notes Egstad. “It probably has the strongest color-correction tools of any major package.”

D2's Linux Movies

D2 had requests for years to make NUKE into a commercial product for use by other studios, and the pressure increased after Apple purchased industry-leader Shake. Studios became concerned when Apple dallied with announcing future Linux support.

“We've founded the D2 Software Company to sell and market NUKE and other applications that currently exist or don't exist within the studio”, says Digital Production and Technology VP Michael Taylor. He continues:

> We have NUKE evaluation sites out in the field. We're providing the latest NUKE 3 version that we use internally. About two years ago when making the decision to do a complete NUKE rewrite incorporating a 3-D into 2-D model, we considered switching to Shake, but decided we had a better program.

Taylor says Linux, Windows and IRIX versions will be available in early 2003. There are no plans yet for Mac OS X. Pricing starts under $10K US, which is comparable to Shake. For students, there will be a free-of-charge or inexpensive version, comparable to the apprentice versions of Maya and Houdini.

Digital compositor Brian Begun describes working on a scene in NUKE for *Star Trek Nemesis*:

> I'm working on a temp, that's a shot that isn't finished—isn't ready for film. We have a production intranet for each show we work on with a web page for each shot. A lot of artists need to share information. Our job system uses Netscape with a lot of HTML forms and a

> server written in Perl. Rather than files in directories, we have links in directories. We can keep files in any directory on any drive anywhere without seeing what drive it is on. This allows our Systems department to juggle our disk space when necessary and to use it as efficiently as possible, without affecting production.

Begun walks us through setting up a typical effect in NUKE—moving the *Enterprise* across a star field:

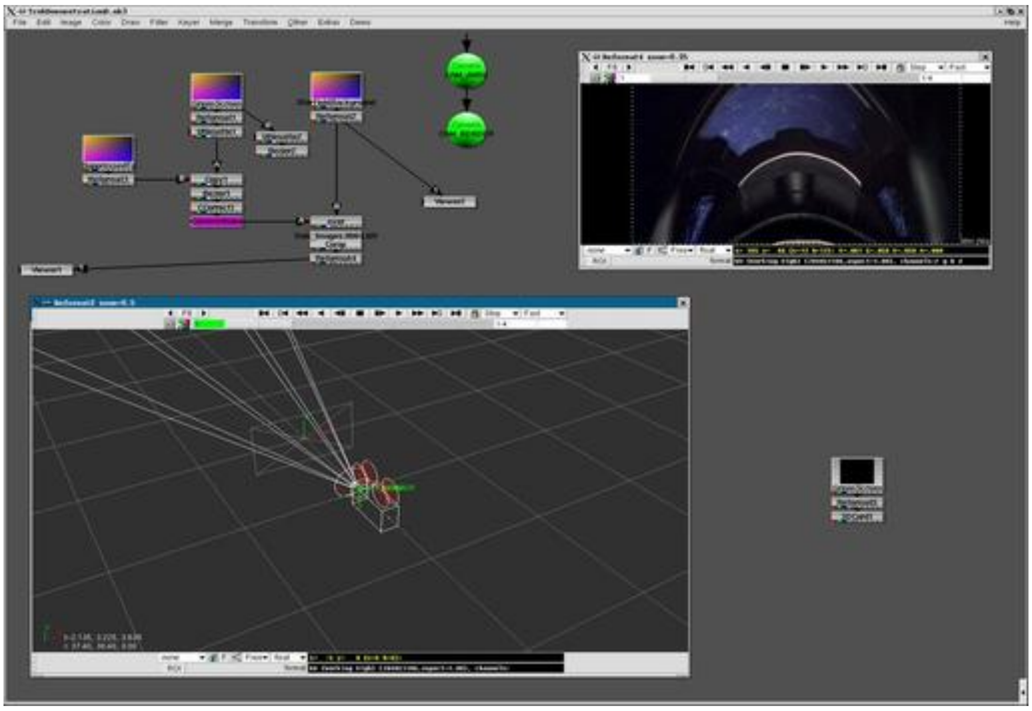


Figure 2. A NUKE window displays an OpenGL 3-D wireframe of the virtual viewing camera.

> Here's *Trek*'s environment. We have a predefined list of variables for each show. Let's say I choose *Star Trek* SS145A:

```
$ job trek                    [sets show variables]
$ shot ss145a                     [sets job variables]
```

> The **cs** command switches to my work directory, in this case work.begun:

```
$ cs
```

> From here, I can go to an image directory that contains elements, parts of composite—or the work directory that contains NUKE scripts and if we do tracking, the in-house Track scripts. The work directory will contain files for NUKE, Flame, Track and Elastic Reality (old but cheap software used for roto and Avid morphing, such as bad frame or wire removal by morphing).

> If I need to create my work directory, I use the jsmk command. Other directories, such as image directories also are created this way. They contain each green screen, full-resolution and scaled-down proxy image, previz and temp comp (which gives the client a rough idea of the shot, but is not necessarily pretty).
>
> The lss command displays files in a more readable format than ls. For example, instead of looking at files like this:

```
test.0001.rgb
test.0002.rgb
test.0003.rgb
```

> Typing **lss** displays files like this:

```
test.%04d.rgb 1-3
```

> Before launching NUKE, I change to the NUKE subdirectory in my work directory:

```
$ cs
$ cd nuke
$ nuke3
```

> When I launch NUKE, it brings up a GUI window, and I choose Image®Read®File and then ss145a.wh to load the foreground (green screen) images. When working on a project, I use both high-resolution images and quarter-resolution proxy images.
>
> The images are Cineon 10-bit log. NUKE itself will convert that to 16-bit float. NUKE is capable of displaying up to ten images in one viewer. By simply entering 1 to 0 on the keyboard, I can have up to ten views.

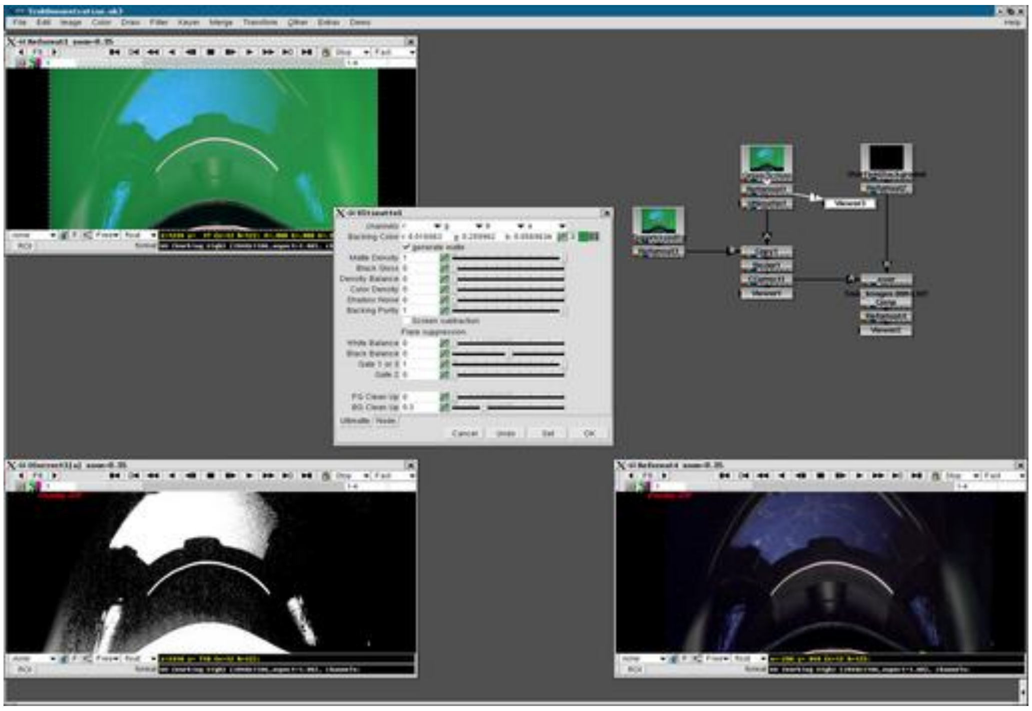


Figure 3. Adjusting a green screen Ultimate composite in NUKE while inserting stars into spacecraft cockpit window.

> Here's a green screen of a cockpit [see Figure 3]. I bring in the background image of stars. When pulling a green screen, you'll typically pull three types of mattes. An edge matte is used to retain all the fine detail present in the photography. A fill or "innie" is used to fill any holes that may occur due to green spill or green material in front of the foreground subject. And, a cleanup or "outtie" matte is used to remove anything that is supposed to be replaced by the background—such as stage lights. To pull these mattes, I'll select a "backing color" in Ultimatte's color picker that best represents the color I want to remove, and that will give me the best matte. After that, I'll make any necessary tweaks, including pulling additional mattes where necessary, or additional cleanup.

Technical Director Jason Iversen is responsible for energy beam effects and debris for *Star Trek*:

> For ships exploding we use as many practical effects as possible. Practicals are faster, even though it takes time to build the model. That may take two guys two months, but it is three people for four to five months to create a 3-D shot. We shoot the explosion at 300 fps slo-mo. It's a big task, and still might not get realism. Some explosions are enhanced with digital debris using Houdini. Some *Enterprise* shots are still real, but not the hull-scraping beauty shots.

As we're talking, one of his SGI machines is being taken away for use on the renderfarm. At D2, workstations are being upgraded to dual-Pentium PCs.

Star Trek work at D2 was previously all done in Houdini on Linux, but most of the Maya artists are on Windows NT because of Maya plugins not being available on Linux. “One of the largest sequences we've got is the avalanche sequence, all in Linux Houdini plus our own internal tool called VoxelB for doing volumetrics”, notes Iversen. He continues:

> The avalanche is a huge powdery trail that is generated in a 3-D sense—not a 2-D cheat. Our voxel compositor VoxelB is a plugin. All of our tools can take in data from Maya or Houdini. We often combine those with our fluid dynamics software to create flowing water.

“Terragen is our terrain-generating program that was used in *Time Machine* for planet shots”, says Iversen. He adds:

> We use it for previz and to create the initial plate for digital painters. Digital actors are all in Maya, primarily on NT. Our pipeline is based on previz rolling into production. All artists do precomposites of our work, then get assigned a compositor to take it to film out.

Although Linux supports popular 3-D packages such as Houdini and Maya, Crow says she feels frustrated by a dearth of Linux paint packages. “There's a depth to Photoshop that Film GIMP doesn't have. Film GIMP isn't mature enough.” Crow says a promising development is Amazon16, a 16-bit paint package that maker Interactive Effects is porting to Linux. Amazon has a long history on IRIX. “It was layer-based before Photoshop, supports user-defined macros, provides 3-D texture paint capabilities, and most importantly, supports HDR formats like Cineon that are critical for film work”, says Crow. “Another promising development is the 32-bit Linux paint package Photogenics by Idruna, currently in beta.”

According to Crow, porting D2's IRIX-based applications to Linux went rapidly, especially with their compositing software NUKE. The Linux conversion at D2 happened in stages, first the renderfarm that performs batch processing of movie effects, then the desktops where artists work. “When Linux was ready for the desktop we were eager to adopt it”, says Crow. “As soon as we got an OS like Linux supporting the features we relied on we were excited to move to it.”

Resources

Robin Rowe (Robin.Rowe@MovieEditor.com) is a partner in motion picture technology company MovieEditor.com and founder of LinuxMovies.org and OpenSourceProgrammers.org.

Archive Index Issue Table of Contents

Advanced search

Zero Copy I: User-Mode Perspective

Dragan Stancevic

Issue #105, January 2003

Explaining what is zero-copy functionality for Linux, why it's useful and where it needs work.

By now almost everyone has heard of so-called zero-copy functionality under Linux, but I often run into people who don't have a full understanding of the subject. Because of this, I decided to write a few articles that dig into the matter a bit deeper, in the hope of unraveling this useful feature. In this article, we take a look at zero copy from a user-mode application point of view, so gory kernel-level details are omitted intentionally.

What Is Zero-Copy?

To better understand the solution to a problem, we first need to understand the problem itself. Let's look at what is involved in the simple procedure of a network server dæmon serving data stored in a file to a client over the network. Here's some sample code:

```
read(file, tmp_buf, len);
write(socket, tmp_buf, len);
```

Looks simple enough; you would think there is not much overhead with only those two system calls. In reality, this couldn't be further from the truth. Behind those two calls, the data has been copied at least four times, and almost as many user/kernel context switches have been performed. (Actually this process is much more complicated, but I wanted to keep it simple). To get a better idea of the process involved, take a look at Figure 1. The top side shows context switches, and the bottom side shows copy operations.

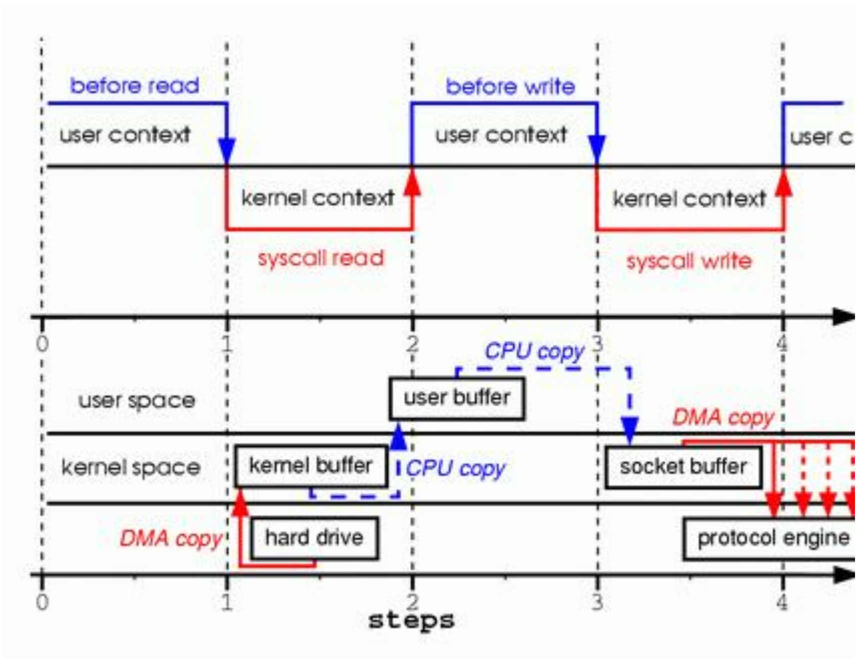


Figure 1. Copying in Two Sample System Calls

Step one: the read system call causes a context switch from user mode to kernel mode. The first copy is performed by the DMA engine, which reads file contents from the disk and stores them into a kernel address space buffer.

Step two: data is copied from the kernel buffer into the user buffer, and the read system call returns. The return from the call caused a context switch from kernel back to user mode. Now the data is stored in the user address space buffer, and it can begin its way down again.

Step three: the write system call causes a context switch from user mode to kernel mode. A third copy is performed to put the data into a kernel address space buffer again. This time, though, the data is put into a different buffer, a buffer that is associated with sockets specifically.

Step four: the write system call returns, creating our fourth context switch. Independently and asynchronously, a fourth copy happens as the DMA engine passes the data from the kernel buffer to the protocol engine. You are probably asking yourself, “What do you mean independently and asynchronously? Wasn't the data transmitted before the call returned?” Call return, in fact, doesn't guarantee transmission; it doesn't even guarantee the start of the transmission. It simply means the Ethernet driver had free descriptors in its queue and has accepted our data for transmission. There could be numerous packets queued before ours. Unless the driver/hardware implements priority rings or queues, data is transmitted on a first-in-first-out basis. (The forked DMA copy in Figure 1 illustrates the fact that the last copy can be delayed).

As you can see, a lot of data duplication is not really necessary to hold things up. Some of the duplication could be eliminated to decrease overhead and increase performance. As a driver developer, I work with hardware that has some pretty advanced features. Some hardware can bypass the main memory altogether and transmit data directly to another device. This feature eliminates a copy in the system memory and is a nice thing to have, but not all hardware supports it. There is also the issue of the data from the disk having to be repackaged for the network, which introduces some complications. To eliminate overhead, we could start by eliminating some of the copying between the kernel and user buffers.

One way to eliminate a copy is to skip calling read and instead call mmap. For example:

```
  tmp_buf = mmap(file, len);
  write(socket, tmp_buf, len);
```

To get a better idea of the process involved, take a look at Figure 2. Context switches remain the same.

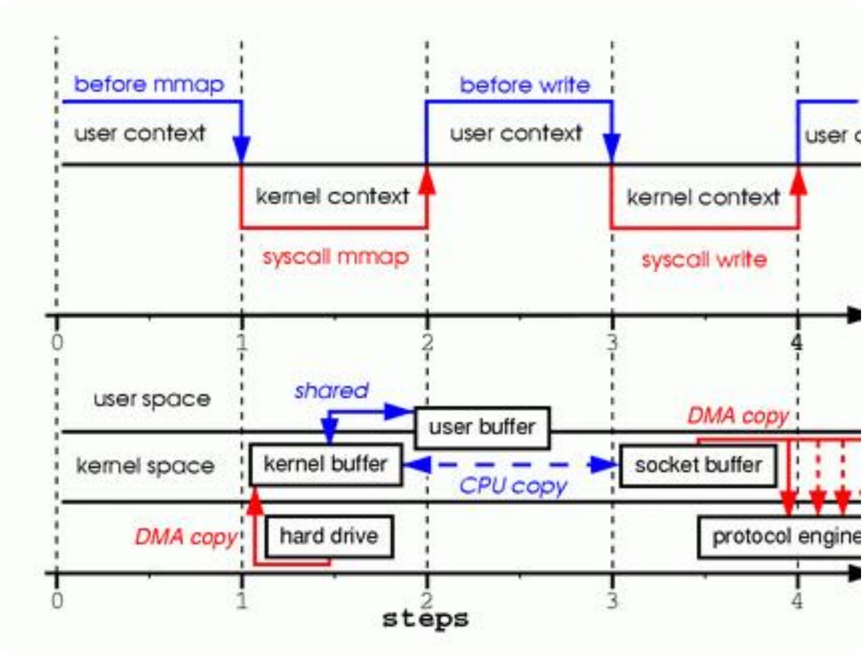


Figure 2. Calling mmap

Step one: the mmap system call causes the file contents to be copied into a kernel buffer by the DMA engine. The buffer is shared then with the user process, without any copy being performed between the kernel and user memory spaces.

Step two: the write system call causes the kernel to copy the data from the original kernel buffers into the kernel buffers associated with sockets.

Step three: the third copy happens as the DMA engine passes the data from the kernel socket buffers to the protocol engine.

By using mmap instead of read, we've cut in half the amount of data the kernel has to copy. This yields reasonably good results when a lot of data is being transmitted. However, this improvement doesn't come without a price; there are hidden pitfalls when using the mmap+write method. You will fall into one of them when you memory map a file and then call write while another process truncates the same file. Your write system call will be interrupted by the bus error signal SIGBUS, because you performed a bad memory access. The default behavior for that signal is to kill the process and dump core—not the most desirable operation for a network server. There are two ways to get around this problem.

The first way is to install a signal handler for the SIGBUS signal, and then simply call return in the handler. By doing this the write system call returns with the number of bytes it wrote before it got interrupted and the errno set to success. Let me point out that this would be a bad solution, one that treats the symptoms and not the cause of the problem. Because SIGBUS signals that something has gone seriously wrong with the process, I would discourage using this as a solution.

The second solution involves file leasing (which is called “opportunistic locking” in Microsoft Windows) from the kernel. This is the correct way to fix this problem. By using leasing on the file descriptor, you take a lease with the kernel on a particular file. You then can request a read/write lease from the kernel. When another process tries to truncate the file you are transmitting, the kernel sends you a real-time signal, the RT_SIGNAL_LEASE signal. It tells you the kernel is breaking your write or read lease on that file. Your write call is interrupted before your program accesses an invalid address and gets killed by the SIGBUS signal. The return value of the write call is the number of bytes written before the interruption, and the errno will be set to success. Here is some sample code that shows how to get a lease from the kernel:

```
if(fcntl(fd, F_SETSIG, RT_SIGNAL_LEASE) == -1) {
    perror("kernel lease set signal");
    return -1;
}
/* l_type can be F_RDLCK F_WRLCK */
if(fcntl(fd, F_SETLEASE, l_type)){
    perror("kernel lease set type");
    return -1;
}
```

You should get your lease before mmaping the file, and break your lease after you are done. This is achieved by calling fcntl F_SETLEASE with the lease type of F_UNLCK.

Sendfile

In kernel version 2.1, the sendfile system call was introduced to simplify the transmission of data over the network and between two local files. Introduction of sendfile not only reduces data copying, it also reduces context switches. Use it like this:

```
  sendfile(socket, file, len);
```

To get a better idea of the process involved, take a look at Figure 3.

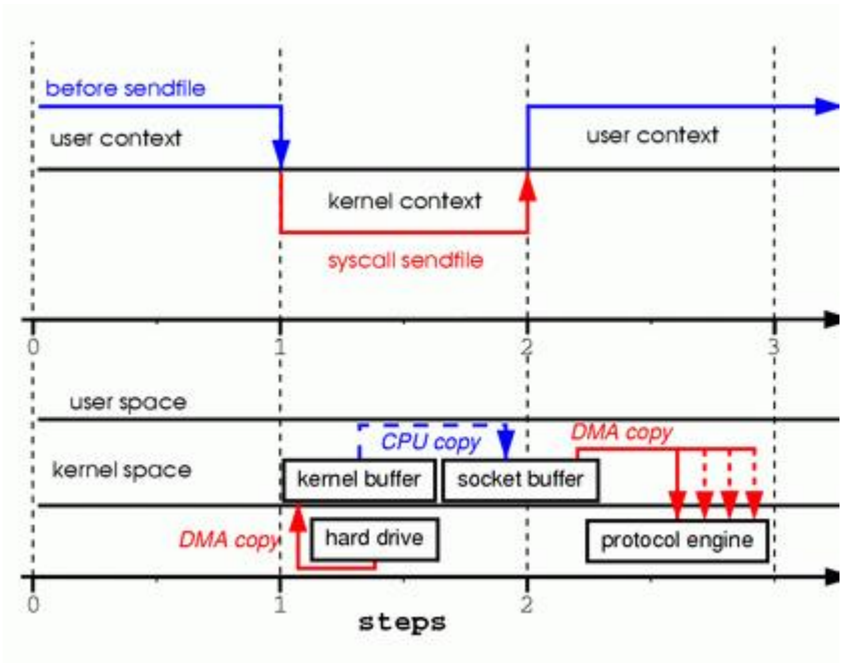


Figure 3. Replacing Read and Write with Sendfile

Step one: the sendfile system call causes the file contents to be copied into a kernel buffer by the DMA engine. Then the data is copied by the kernel into the kernel buffer associated with sockets.

Step two: the third copy happens as the DMA engine passes the data from the kernel socket buffers to the protocol engine.

You are probably wondering what happens if another process truncates the file we are transmitting with the sendfile system call. If we don't register any signal handlers, the sendfile call simply returns with the number of bytes it transferred before it got interrupted, and the errno will be set to success.

If we get a lease from the kernel on the file before we call sendfile, however, the behavior and the return status are exactly the same. We also get the RT_SIGNAL_LEASE signal before the sendfile call returns.

So far, we have been able to avoid having the kernel make several copies, but we are still left with one copy. Can that be avoided too? Absolutely, with a little help from the hardware. To eliminate all the data duplication done by the kernel, we need a network interface that supports gather operations. This simply means that data awaiting transmission doesn't need to be in consecutive memory; it can be scattered through various memory locations. In kernel version 2.4, the socket buffer descriptor was modified to accommodate those requirements—what is known as zero copy under Linux. This approach not only reduces multiple context switches, it also eliminates data duplication done by the processor. For user-level applications nothing has changed, so the code still looks like this:

```
sendfile(socket, file, len);
```

To get a better idea of the process involved, take a look at Figure 4.

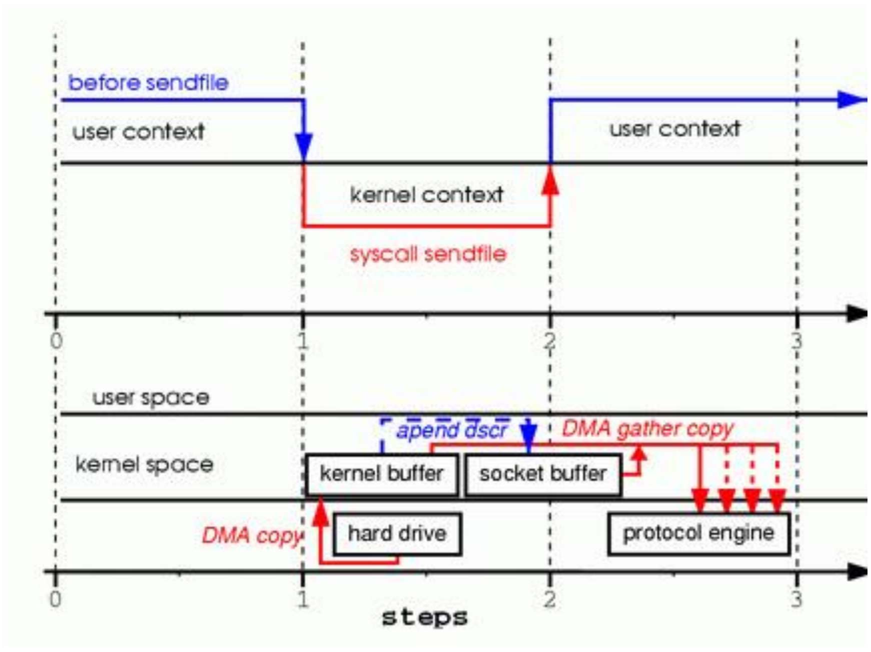


Figure 4. Hardware that supports gather can assemble data from multiple memory locations, eliminating another copy.

Step one: the sendfile system call causes the file contents to be copied into a kernel buffer by the DMA engine.

Step two: no data is copied into the socket buffer. Instead, only descriptors with information about the whereabouts and length of the data are appended to the socket buffer. The DMA engine passes data directly from the kernel buffer to the protocol engine, thus eliminating the remaining final copy.

Because data still is actually copied from the disk to the memory and from the memory to the wire, some might argue this is not a true zero copy. This is zero copy from the operating system standpoint, though, because the data is not duplicated between kernel buffers. When using zero copy, other performance benefits can be had besides copy avoidance, such as fewer context switches, less CPU data cache pollution and no CPU checksum calculations.

Now that we know what zero copy is, let's put theory into practice and write some code. You can download the full source code from www.xalien.org/articles/source/sfl-src.tgz. To unpack the source code, type **tar -zxvf sfl-src.tgz** at the prompt. To compile the code and create the random data file data.bin, run **make**.

Looking at the code starting with header files:

```
/* sfl.c sendfile example program
Dragan Stancevic <
header name                 function / variable
-------------------------------------------------*/
#include <stdio.h>          /* printf, perror */
#include <fcntl.h>          /* open */
#include <unistd.h>         /* close */
#include <errno.h>          /* errno */
#include <string.h>         /* memset */
#include <sys/socket.h>     /* socket */
#include <netinet/in.h>     /* sockaddr_in */
#include <sys/sendfile.h>   /* sendfile */
#include <arpa/inet.h>      /* inet_addr */
#define BUFF_SIZE (10*1024) /* size of the tmp
                               buffer */
```

Besides the regular <sys/socket.h> and <netinet/in.h> required for basic socket operation, we need a prototype definition of the sendfile system call. This can be found in the <sys/sendfile.h> server flag:

```
/* are we sending or receiving */
if(argv[1][0] == 's') is_server++;
/* open descriptors */
sd = socket(PF_INET, SOCK_STREAM, 0);
if(is_server) fd = open("data.bin", O_RDONLY);
```

The same program can act as either a server/sender or a client/receiver. We have to check one of the command-prompt parameters, and then set the flag is_server to run in sender mode. We also open a stream socket of the INET protocol family. As part of running in server mode we need some type of data to transmit to a client, so we open our data file. We are using the system call sendfile to transmit data, so we don't have to read the actual contents of the file and store it in our program memory buffer. Here's the server address:

```
/* clear the memory */
memset(&sa, 0, sizeof(struct sockaddr_in));
/* initialize structure */
sa.sin_family = PF_INET;
sa.sin_port = htons(1033);
sa.sin_addr.s_addr = inet_addr(argv[2]);
```

We clear the server address structure and assign the protocol family, port and IP address of the server. The address of the server is passed as a command-line parameter. The port number is hard coded to unassigned port 1033. This port number was chosen because it is above the port range requiring root access to the system.

Here is the server execution branch:

```
if(is_server){
    int client; /* new client socket */
    printf("Server binding to [%s]\n", argv[2]);
    if(bind(sd, (struct sockaddr *)&sa,
                      sizeof(sa)) < 0){
        perror("bind");
        exit(errno);
    }
```

As a server, we need to assign an address to our socket descriptor. This is achieved by the system call bind, which assigns the socket descriptor (sd) a server address (sa):

```
if(listen(sd,1) < 0){
    perror("listen");
    exit(errno);
}
```

Because we are using a stream socket, we have to advertise our willingness to accept incoming connections and set the connection queue size. I've set the backlog queue to 1, but it is common to set the backlog a bit higher for established connections waiting to be accepted. In older versions of the kernel, the backlog queue was used to prevent syn flood attacks. Because the system call listen changed to set parameters for only established connections, the backlog queue feature has been deprecated for this call. The kernel parameter tcp_max_syn_backlog has taken over the role of protecting the system from syn flood attacks:

```
if((client = accept(sd, NULL, NULL)) < 0){
    perror("accept");
    exit(errno);
}
```

The system call accept creates a new connected socket from the first connection request on the pending connections queue. The return value from the call is a descriptor for a newly created connection; the socket is now ready for read, write or poll/select system calls:

```
if((cnt = sendfile(client,fd,&off,
                          BUFF_SIZE)) < 0){
    perror("sendfile");
    exit(errno);
}
printf("Server sent %d bytes.\n", cnt);
close(client);
```

A connection is established on the client socket descriptor, so we can start transmitting data to the remote system. We do this by calling the sendfile system call, which is prototyped under Linux in the following manner:

```
extern ssize_t
sendfile (int __out_fd, int __in_fd, off_t *offset,
          size_t __count) __THROW;
```

The first two parameters are file descriptors. The third parameter points to an offset from which sendfile should start sending data. The fourth parameter is the number of bytes we want to transmit. In order for the sendfile transmit to use zero-copy functionality, you need memory gather operation support from your networking card. You also need checksum capabilities for protocols that implement checksums, such as TCP or UDP. If your NIC is outdated and doesn't support those features, you still can use sendfile to transmit files. The difference is the kernel will merge the buffers before transmitting them.

Portability Issues

One of the problems with the sendfile system call, in general, is the lack of a standard implementation, as there is for the open system call. Sendfile implementations in Linux, Solaris or HP-UX are quite different. This poses a problem for developers who wish to use zero copy in their network data transmission code.

One of the implementation differences is Linux provides a sendfile that defines an interface for transmitting data between two file descriptors (file-to-file) and (file-to-socket). HP-UX and Solaris, on the other hand, can be used only for file-to-socket submissions.

The second difference is Linux doesn't implement vectored transfers. Solaris sendfile and HP-UX sendfile have extra parameters that eliminate overhead associated with prepending headers to the data being transmitted.

Looking Ahead

The implementation of zero copy under Linux is far from finished and is likely to change in the near future. More functionality should be added. For example, the sendfile call doesn't support vectored transfers, and servers such as Samba and Apache have to use multiple sendfile calls with the TCP_CORK flag set. This flag tells the system more data is coming through in the next sendfile calls. TCP_CORK also is incompatible with TCP_NODELAY and is used when we want to prepend or append headers to the data. This is a perfect example of where a vectored call would eliminate the need for multiple sendfile calls and delays mandated by the current implementation.

One rather unpleasant limitation in the current sendfile is it cannot be used when transferring files greater than 2GB. Files of such size are not all that uncommon today, and it's rather disappointing having to duplicate all that data on its way out. Because both sendfile and mmap methods are unusable in this case, a sendfile64 would be really handy in a future kernel version.

Conclusion

Despite some drawbacks, zero-copy sendfile is a useful feature, and I hope you have found this article informative enough to start using it in your programs. If you have a more in-depth interest in the subject, keep an eye out for my second article, titled “Zero Copy II: Kernel Perspective”, where I will dig a bit more into the kernel internals of zero copy.

Further Information

email: visitor@xalien.org

Dragan Stancevic is a kernel and hardware bring-up engineer in his late twenties. He is a software engineer by profession but has a deep interest in applied physics and has been known to play with extremely high voltages in his free time.

Compiling Java with GCJ

Per Bothner

Issue #105, January 2003

Although Java isn't a popular choice for free projects, GJC can make it a viable option.

Java has not become as pervasive as the original hype suggested, but it is a popular language, used a lot for in-house and server-side development and other applications. Java has less mind-share in the free software world, although many projects are now using it. Examples of free projects using Java include Jakarta from the Apache Foundation (jakarta.apache.org), various XML tools from W3C (www.w3.org) and Freenet (freenet.sourceforge.net). See also the FSF's Java page (www.gnu.org/software/java).

One reason relatively few projects use Java has been the real or perceived lack of quality, free implementations of Java. Two free Java implementations, however, have been around since the early days of Java. One is Kaffe (www.kaffe.org), originally written by Tim Wilkinson and still developed by the company he cofounded, Transvirtual. The other is GCJ (the GNU Compiler for the Java language), which I started in 1996 at Cygnus Solutions (and which this article discusses). GCJ has been fully integrated and supported as a GCC language since GCC version 3.0.

Traditional Java Implementation Model

The traditional way to implement Java is a two-step process: a translation phase and an execution phase. (In this respect Java is like C.) A Java program is compiled by javac, which produces one or more files with the extension .class. Each such file is the binary representation of the information in a single class, including the expressions and statements of the class' methods. All of these have been translated into bytecode, which is basically the instruction set for a virtual, stack-based computer. (Because some chips also have a Java bytecode instruction set, it also can be a real instruction set.)

The execution phase is handled by a Java Virtual Machine (JVM) that reads in and executes the .class files. Sun's version is called plain “java”. Think of the JVM as a simulator for a machine whose instruction set is Java bytecodes.

Using an interpreter (simulator) adds quite a bit of execution overhead. A common solution for high-performance JVMs is to use dynamic translation or just-in-time (JIT) compilers. In that case, the runtime system will notice a method has been called enough times to make it worthwhile to generate machine code for that method on the fly. Future calls to the method will execute the machine code directly.

A problem with JITs is startup overhead. It takes time to compile a method, especially if you want to do any optimization, and this compilation is done each time the application is run. If you decide to compile only the methods most often executed, then you have the overhead of measuring those. Another problem is that a good JIT is complex and takes up a fair bit of space (plus the generated code needs space, which may be on top of the space used by the original bytecode). Little of this space can be in shared memory.

Traditional Java implementation techniques also do not interoperate well with other languages. Applications are deployed differently (a Java Archive .jar file, rather than an executable); they require a big runtime system, and calling between Java and C/C++ is slow and inconvenient.

The GCJ Solution: Ahead-of-Time Compilation

The approach of the GCJ Project is radically traditional. We view Java as simply another programming language and implement it the way we implement other compiled languages. As Cygnus had been long involved with GCC, which was already being used to compile a number of different programming languages (C, C++, Pascal, Ada, Modula2, Fortran, Chill), it made sense to think about compiling Java to native code using GCC.

On the whole, compiling a Java program is actually much simpler than compiling a C++ program, because Java has no templates and no preprocessor. The type system, object model and exception-handling model are also simpler. In order to compile a Java program, the program basically is represented as an abstract syntax tree, using the same data structure GCC uses for all of its languages. For each Java construct, we use the same internal representation as the equivalent C++ would use, and GCC takes care of the rest.

GCJ can then make use of all the optimizations and tools already built for the GNU tools. Examples of optimizations are common sub-expression elimination, strength reduction, loop optimization and register allocation. Additionally, GCJ can do more sophisticated and time-consuming optimizations than a just-in-time compiler can. Some people argue, however, that a JIT can do more tailored and adaptive optimizations (for example, change the code depending on actual execution). In fact, Sun's HotSpot technology is based on this premise, and it certainly does an impressive job. Truthfully, running a program compiled by GCJ is not always noticeably faster than running it on a JIT-based Java implementation; sometimes it even may be slower, but that usually is because we have not had time to implement Java-specific optimizations and tuning in GCJ, rather than any inherent advantage of HotSpot technology. GCJ is often significantly faster than alternative JVMs, and it is getting faster as people improve it.

A big advantage of GCJ is startup speed and modest memory usage. Originally, people claimed that bytecode was more space-efficient than native instruction sets. This is true to some extent, but remember that about half the space in a .class file is taken up by symbolic (non-instruction) information. These symbols are duplicated for each .class file, while ELF executables or libraries can do much more sharing. But where bytecodes really lose out to native code is in terms of memory inside a JVM with a JIT. Starting up Sun's JVM and JIT compiling and applications' classes take a huge amount of time and memory. For example, Sun's IDE Forte for Java (available in the NetBeans open-source version) is huge. Starting up NetBeans takes 74MB (as reported by the **top** command) before you actually start doing anything. The amount of main memory used by Java applications complicates their deployment. An illustration is JEmacs (JEmacs.sourceforge.net), a (not very active) project of mine to implement Emacs in Java using Swing (and Kawa, discussed below, for Emacs Lisp support). Starting up a simple editor window using Sun's JDK1.3.1 takes 26MB (according to top). XEmacs, in contrast, takes 8MB.

Running the Kawa test suite using GCJ vs. JDK1.3.1, GCJ is about twice as fast, causes about half the page faults (according to the **time** command) and uses about 25% less memory (according to top). The test suite is a script that starts the Java environment multiple times and runs too many different things for a JIT to help (which penalizes JDK). It also loads Scheme code interactively, so GCJ has to run it using its interpreter (which penalizes GCJ). This experiment is not a real benchmark, but it does indicate that even in its current status you can get improved performance using GCJ. (As always, if you are concerned about performance, run your own benchmark based on your expected job mix.)

GCJ has other advantages, such as debugging with GDB and interfacing with C/C++ (mentioned below). Finally, GCJ is free software, based on the industry-standard GCC, allowing it to be freely modified, ported and distributed.

Some have complained that ahead-of-time compilation loses the big write-once, run-anywhere portability advantage of bytecodes. However, that argument ignores the distinction between distribution and installation. We do not propose native executables as a distribution format, expect perhaps as prebuilt packages (e.g., RPMs) for a particular architecture. You still can use Java bytecodes as a distribution format, even though they don't have any major advantages over Java source code. (Java source code tends to have fewer portability problems than C or C++ source.) We suggest that when you install a Java application, you should compile it to native code if it isn't already so compiled.

Compiling a Java Program with GCJ

Using GCC to run a Java program is familiar to anyone who has used it for C or C++ programs. To compile the Java program MyJavaProg.java, type:

```
gcj -c -g -O MyJavaProg.java
```

To link it, use the command:

```
gcj --main=MyJavaProg -o MyJavaProg MyJavaProg.o
```

This is just like compiling a C++ program mycxxprog.cc:

```
g++ -c -g -O mycxxprog.cc
```

and then linking to create an executable mycxxprog:

```
g++ -o mycxxprog mycxxprog.o
```

The only new aspect is the option --main=MyJavaProg. This is needed because it is common to write Java classes containing a main method that can be used for testing or small utilities. Thus, if you link a bunch of Java-compiled classes together, there may be many main methods, and you need to tell the linker which one should be called when the application starts.

You also have the option of compiling a set of Java classes into a shared library (.so file). In fact, the GCJ runtime system is compiled to a .so file. While the details of this belong in another article, if you are curious you can look at the Makefiles of Kawa (discussed below) to see how this works.

Features and Limitations of GCJ

GCJ is not only a compiler. It is intended to be a complete Java environment with features similar to Sun's JDK. If you specify the -C option to gcj it will compile to standard .class files. Specifically, the goal is that **gcj -C** should be a plugin replacement for Sun's javac command.

GCJ comes with a bytecode interpreter (contributed by Kresten Krab Thorup) and has a fully functional ClassLoader. The standalone gij program works as a plugin replacement for Sun's java command.

GCJ works with libgcj, which is included in GCC 3.0. This runtime library includes the core runtime support, Hans Boehm's well-regarded conservative garbage collector, the bytecode interpreter and a large library of classes. For legal and technical reasons, GCJ cannot ship Sun's class library, so it has its own. The GNU Classpath Project now uses the same license and FSF copyright that libgcj and libstdc++ use, and classes are being merged between the two projects. We use the GPL but with the special exception that if you link libgcj with other files to produce an executable, this does not by itself cause the executable to be compiled by the GPL. Thus, even proprietary programs can be linked with the standard C++ or Java; runtime libraries.

The libgcj library includes most of the standard Java classes needed to run non-GUI applications, including all or most of the classes in the java.lang, java.io, java.util, java.net, java.security, java.sql and java.math packages. The major missing components are classes for doing graphics using AWT or Swing. Most of the higher-level AWT classes are implemented, but the lower-level peer classes are not complete enough to be useful. Volunteers are needed to help out.

Interoperating with C/C++

Although you can do a lot using Java, sometimes you want to call libraries written in another language. This could be because you need to access low-level code that cannot be written to Java, an existing library provides functionality that you need and don't want to rewrite, or you need to do low-level performance hacks for speed. You can do all of these by declaring some Java methods to be native and, instead of writing a method body, provide an implementation in some other language. In 1997, Sun released the Java Native Interface (JNI), which is a standard for writing native methods in either C or C++. The main goal of JNI is portability in the sense that native methods written for one Java implementation should work with another Java implementation, without recompiling. This was designed for a closed-source, distributed-binaries world and is less valuable in a free-software context, especially because you do have to recompile if you change chips or the OS type.

To ensure JNI's portability, everything is done indirectly using a table of functions. This makes JNI very slow. Even worse, writing all these functions and following all the rules is tedious and error-prone. Although GCJ does support JNI, it also provides an alternative. The Compiled Native Interface (CNI, which could also stand for Cygnus Native Interface) is based on the idea that Java is basically a subset of C++ and that GCC uses the same calling convention for C++

and Java. So, what could be more natural than being able to write Java methods using C++ and using standard C++ syntax for access to Java fields and calls to Java methods? Because they use the same calling conventions and data layout, no conversion or magic glue is needed between C++ and Java.

Examples of CNI and JNI will have to wait for a future article. The GCJ manual (gcc.gnu.org/onlinedocs/gcj) covers CNI fairly well, and the libgcj sources include many examples.

Kawa: Compiling Scheme to Native via Java

Java bytecodes are a fairly direct encoding of Java programs not really designed for anything else. However, they have been used to encode programs written in other languages. See grunge.cs.tu-berlin.de/~tolk/vmlanguages.html for a list of other programming languages implemented on top of Java. Most of these are interpreters, but a few actually compile to bytecode. The former could use GCJ as is; the latter potentially can use GCJ to compile to native code.

One such compiler is Kawa, which I have been developing since 1996. Kawa is both a toolkit for implementing languages using Java and an implementation of the Scheme programming language. You can build and run Kawa using GCJ without needing any non-free software. The Kawa home page (www.gnu.org/software/kawa) has instructions for downloading and building Kawa with GCJ.

You can use Kawa in interactive mode. Here, we first define the factorial function and then call it:

```
$ kawa
#|kawa:1|# (define (factorial x)
#|(---:2|#  (if (< x 2) x (* x (factorial (- x 1)))))
#|kawa:3|# (factorial 30)
265252859812191058636308480000000
```

An interesting thing to note is the factorial function actually gets compiled by Kawa to bytecode and is immediately loaded as a new class. This process uses Java's ClassLoader mechanism to define a new class at runtime for a byte array containing the bytecodes for the class. The methods of the new class are interpreted by GCJ's bytecode interpreter.

Of course, it is usually more convenient to put the code in a file:

```
$ cat > factorial.scm
(define (factorial x)
(if (< x 2) x (* x (factorial (- x 1)))))
(format #t "Factorial ~d is ~d.~%~!" 30 (factorial 30))
^D
$ kawa -f factorial.scm
Factorial 30 is 265252859812191058636308480000000.
```

You can increase the performance of Scheme code by using Kawa to compile it ahead of time, creating one or more .class files:

```
$ kawa --main -C factorial.scm
(compiling factorial.scm)
```

You can then load the compiled file:

```
$ kawa -f factorial.class
Factorial 30 is 265252859812191058636308480000000.
```

To compile the class file to native code, you can use gckawa, a script that sets up appropriate environment variables (LD_LIBRARY_PATH and CLASSPATH) and calls gcj:

```
$ gckawa -o factorial
--main=factorial -g -O factorial*.class
```

Using the wildcard in factorial*.class is not needed in this case, but it is a good idea in case Kawa needs to generate multiple .class files.

Then, you can execute the resulting factorial program, which is a normal GNU/ Linux ELF executable. It links with the shared libraries libgcj.so (the GCJ runtime library) and libkawa.so (the Kawa runtime library).

The same approach can be used for other languages. For example, I am currently working on implementing XQuery, W3C's new XML-query language, using Kawa.

Other applications that have been built with GCJ include Apache modules, GNU-Paperclips and Jigsaw.

Conclusion

GCJ has seen a lot of activity recently and is a solid platform for many tasks. We hope that you consider Java for your free software project, using GCJ as your preferred Java implementation and that some of you will help make GCJ even better.

email: per@bothner.com

Per Bothner (www.bothner.com/per) has worked on GNU software since the 1980s. At Cygnus he was technical leader of the GCJ Project. He is currently an independent consultant.

Apache Talking IPv6

Ibrahim Haddad

David Gordon

Issue #105, January 2003

A technical tutorial on setting up Apache to serve HTTP requests over IPv6 and some preliminary benchmarking results.

IP version 6 (IPv6) is the newest version of the Internet Protocol, designed by the IETF as a successor to IP version 4 (IPv4). In this article, we address the case of running the Apache web server over IPv6.

The Apache Web Server

According to the Netcraft web server survey, Apache has been the most popular web server on the Internet since April 1996. Currently, over 56% of all web servers run Apache. These numbers come as no surprise due to Apache's portability over multiple platforms, reliability, robustness, configurability, and the fact that it is free and well documented.

Apache 1.3 has established itself as a high-performance web server. However, with the evolution of requirements imposed on web servers, new functionalities, such as higher reliability, security and performance and scalability are required. In response, Apache continued its drive to satisfy these new requirements with version 2.0, promising a more robust and faster web server with new and enhanced functionalities.

Apache 2.0 offers numerous performance improvements (the subject of an article I wrote for the *Linux Journal* web site, www.linuxjournal.com/articles/4559). However, as far as this article is concerned, one of the new features is support for IPv6. With the 2.0 version, if you run Apache on systems where IPv6 support exists, Apache gets IPv6 listening sockets by default. Additionally, the Listen, NameVirtualHost and VirtualHost directives also support IPv6 numeric address strings.

In the following sections, we demonstrate how to add support for IPv6 in the Linux kernel and then show how to install the latest Apache version and run it with IPv6 support. We also run some benchmarking tests to compare the performance of the same server servicing requests using IPv4 and IPv6.

Supporting IPv6 at the Kernel Level

In this section, we briefly describe how to enable IPv6 in the Linux kernel, a prerequisite to enable IPv6 HTTP requests. You may already have IPv6 support in your kernel [see page 64]. To add it, the first step is to download a stable Linux kernel and uncompress it. For our testing we downloaded kernel 2.4.8 from kernel.org.

We configured the kernel to enable support for IPv6. There are two options you need to enable. In the Code Maturity Level section, you need to enable **"Prompt for development and/or incomplete code/drivers"**.

And in the Networking Options section, you need to enable **"IPv6 Protocol (EXPERIMENTAL)"**.

You can choose to compile the IPv6 support within the kernel or as a separate module, depending on your preference. After that, compile and install the kernel and modules as usual, then reboot with IPv6 support.

See the Kernel HOWTO for details on the kernel build and install process (www.tldp.org/HOWTO/Kernel-HOWTO.html).

Downloading and Installing Apache

Now that your kernel supports IPv6, you are ready to install Apache and run it with IPv6 support. First, download the latest Apache distribution from www.apache.org/dist/httpd into /tmp. For illustrative purposes we use Apache 2.0.16; however, the same procedure applies to newer versions. Extract the source:

```
cd /tmp
tar xzvf httpd-2_0_16-beta.tar.gz
```

which will create a new directory, httpd-2_0_16, containing the source code. To configure Apache for your platform and specific requirements, use the configure script included in the root directory of the distribution. You should **cd** into the httpd-2_0_16 directory and type **./configure** at the shell prompt. If you want to know all the options you could pass to the configuration script, type **./ configure --help**. To change the default options, the configure script accepts a variety of variables and command-line options. One of the options is the location prefix where Apache is to be installed. By default, Apache will install into /usr/local/apache. If you want to install Apache in /usr/local/apache-2_0_16, for instance, type:

```
   ./configure --prefix=/usr/local/apache-2_0_16
```

The same applies for other options. When **configure** is run it will take a few seconds to test for the availability of features on your system and build Makefiles that will be used to compile the server. After running the configuration script, you can build the various parts that form the Apache package by running **make && make install**. This will compile Apache and install all the files in /usr/local/apache-2_0_16. The next step is to customize the Apache HTTP server by editing the httpd.conf file under /usr/local/apache-2_0_16/conf or, if you have used a different prefix, under *prefix*/conf/:

```
   vi prefix/conf/httpd.conf
```

The must-edit configuration variables include ServerAdmin, your e-mail address (so you will be alerted in case the server has problems); ServerName, the name or IP of the server; and Port, the port to which the server should listen.

The httpd.conf comes with a lot of explanations, and it is easy to read, understand and customize. However, if you need more details on the configuration directives, you can read the Apache manual either locally under the docs/manual/ in the Apache installation directory or on the Web at httpd.apache.org/docs.

Starting the Server

Now you have compiled, installed and customized the Apache configuration. As we mentioned previously, IPv6 support is now included within the Apache source code, so you do not need to do any special configuration to activate it. To start the server, you can use the Apache Control Script that is designed to allow an easy command-line interface to control Apache. Using it you can start the server, stop it, restart it, check its status and do a configuration syntax test. Therefore, to start the server you would type **apachectl start**, and then you should be able to request documents via http://ServerName or http://localhost/.

Serving Documents over IPv6

With the advent of 128-bit addressing comes the pains of typing long IP addresses. The correct syntax of an IPv6 address is eight fields of four hexadecimal characters separated by colons, totaling 128 bits. Note that in URLs, port numbers can be appended to addresses following a colon, like this: 149.76.14.14:80. Since IPv6 addresses already make use of the colon within the address itself, the address in an IPv6 URL is enclosed in square brackets and the colon and port appended after the closing square bracket, like this: [3ffe:200:8:1000:250:bbff:fe00:25]:80.

Not all web browsers are capable of parsing IPv6 addresses. Netscape Navigator 6.x and Mozilla are the two tested browsers that do support IPv6 addresses. We believe that although their support for IPv6 is not yet mature (as they sometimes freeze and require killing the browser process or restarting it), both of them do allow successful parsing of aliased hostnames. Aliased hostnames are hostnames assigned to a unique IP address in /etc/hosts file. For instance, we can define aliases for IPv6 address by editing /etc/hosts as follows:

```
::1                             node02-v6-localhost
3ffe:200:8:1000:250:bbff:fe00:25 node02-v6
```

When we start Netscape or Mozilla, we type in the defined aliases as the URL and use these aliases to request a web page over IPv6, such as http://node2-v6/ or http://node2-v6-localhost/.

Benchmarking Results

To benchmark the performance of Apache, we followed the above steps to install the latest kernel with IPv6 support and install Apache 2.0.16. The machine is a 1U Celeron 500MHz rackmount unit with 256MB of RAM. It runs Red Hat 7.0. As for benchmarking, we used ApacheBench, a tool that comes free with the Apache web server.

We ran two tests: Apache 2.0.16 over IPv4 and then Apache 2.0.16 over IPv6. For comparison's sake, we ran the same tests on another machine with the same setup, except with Apache 1.3.19, to be able to compare the performance and support for IPv6 in the two versions. To enable IPv6 with Apache 1.3.19, we downloaded the IPv6 Apache patch from the Kame Project web site and applied it to the 1.3.19 source tree. Next, we ran the configure script:

```
./configure --enable-rule=INET6
```

and enabled the INET6 option. Lastly, we did a **make && make install**, which compiled and installed Apache 1.3.19 with IPv6 support from Kame.

Table 1 shows the results of the benchmarking tests. Each of the benchmarks was the result of 1,000 requests with a concurrency level of 1. There were no failed requests or write errors.

Table 1. Benchmarking Results

There are few remarks regarding the results. Apache 1.3.19 was able to serve more requests per second than Apache 2.0.16. In the tested versions of Apache, we had fewer requests per second in IPv6 compared to IPv4; this may be due to the fact that the IPv6 code added to Apache has not been tested and debugged thoroughly. On the other hand, if we examine the transfer rate of Apache 1.3.19, we notice that it is much higher than the transfer rate of Apache 2.0.16. This still needs to be investigated.

At the Ericsson Open Architecture Research Lab, we currently have a benchmarking environment to test the performance of our Linux clusters and application servers (including Apache, Tomcat and Jigsaw web servers). However, the environment and the tests are designed to work with IPv4, not IPv6. Our plan is to port the environment to generate IPv6 HTTP requests in order to test the performance of Apache (and other web servers) with IPv6 under heavy load. We currently have a total of 100 1U rackmount units (a mix of Celeron 500 and Pentium III machines with 256MB and 512 MB of RAM) that soon will be generating IPv6 traffic with one node collecting and compiling the results. We will publish our results as soon as the work is completed. This will be more comprehensive than the preliminary testing that was done for the purposes of this article.

Conclusion

IPv6 is becoming a reality. For the next few years, we need to be able to support both IPv4 and IPv6 on our application servers before the complete transition to IPv6 occurs. The developers of Apache are aware of this, and they have already built in support for IPv6 in the Apache code. Previously, the support for IPv6 was in the form of a downloadable patch (from the Kame Project, for instance) that you would apply to the source tree.

As the benchmarking results suggest, serving documents over IPv6 is slightly slower (in terms of requests/second) than serving them over IPv4; however, this is understandable because the IPv6 support is still in its early stages. As the development advances, Apache is expected to reach and exceed the level of performance it achieved with IPv4 to remain the preferred web server in the world.

Acknowledgements

Ericsson Research (Open Architecture Research Lab) for supporting our work with Linux and open-source software and for approving the publication of this article.

Resources

Ibrahim Haddad (Ibrahim.Haddad@Ericsson.com)is a researcher at the Ericsson Research Open Architecture Lab in Montréal, Canada, where he is primarily involved in researching carrier-class server nodes for real-time all-IP networks. He is currently a Dr.Sc. Candidate at Concordia University.

David Gordon (davidgordonca@yahoo.ca) has completed his co-op term at Ericsson Research Canada as part of the team working to support IPv6 on near telecom-grade Linux clusters. He is currently a Computer Science student at the University of Sherbrooke.

Understand Quicksort with DDD

Adam Monsen

Issue #105, January 2003

Two fundamental steps in your programming education are to learn a good debugger and to understand Quicksort. Let's do them at the same time.

Sorting is one of the most common functions performed by a computer, and Quicksort is one of the most efficient ways to do it. This article demonstrates the usefulness of a graphical debugger for learning how Quicksort works.

DDD (Data Display Debugger) is a free, visual front end that can control many popular debuggers. This article uses DDD to work through a simple C implementation of Quicksort.

First, find a copy of DDD and install it. Binary and source packages are available for RPM-based distributions at rpmfind.net, and Debian packages are available at debian.org. This article was written using DDD version 3.3.1 on Red Hat 7.3. This article also makes the following assumptions: 1) you have a GNU/Linux-enhanced computer and it is plugged in; 2) you know basic C concepts including arrays, loops and recursion; and 3) you have a capable C compiler, such as GNU's GCC. Even if you don't know anything about programming, try stepping through the code anyway. It's good for you.

A Simplistic Quicksort

CAR Hoare described the Quicksort algorithm in a much-cited 1962 paper, and it is still in common use 40 years later. The divide-and-conquer approach of Quicksort is probably where it got the prefix quick; by segregating smaller elements from larger elements it eliminates the need for many comparisons. In contrast, a selection sort compares every element to every other element. This is not to say that Quicksort is always faster or that it's the best way to sort; it's simply cool to know. The implementation in this article is not an optimized or extensible quicksort. It only works on integer arrays.

This code was largely borrowed from *The Practice of Programming* by Brian W. Kernighan and Rob Pike:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/* from: The Practice of Programming (pp: 32-34)
 *   by: Brian W. Kernighan and Rob Pike
 */
/* swap: interchange v[i] and v[j] */
void swap( int v[], int i, int j )
{
  int tmp;
  tmp = v[i];
  v[i] = v[j];
  v[j] = tmp;
}
/* quicksort: sort v[0]..v[n-1] into increasing
 * order
 */
void quicksort( int v[], int n )
{
  int i = 0, last = 0;
  if ( n <= 1 )              /* nothing to do */
    return;
  swap( v, 0, rand() % n );  /* move pivot elem
                                to v[0] */
  for ( i = 1; i < n; i++ )  /* partition */
    if ( v[i] < v[0] )
      swap( v, ++last, i );
  swap( v, 0, last );        /* restore pivot */
  quicksort( v, last );      /* sort smaller
                                values */
  quicksort( v+last+1, n-last-1 );  /* sort larger
                                       values */
}
void print_array( const int array[], int elems )
{
  int i;
  printf("{ ");
  for ( i = 0; i < elems; i++ )
    printf( "%d ", array[i] );
  printf("}\n");
}
#define NUM 9
int main( void )
{
  int arr[NUM] = { 6, 12, 4, 18, 3,
                  27, 16, 15, 19 };
  /* commented out to aid in predictability
   * srand( (unsigned int) time(NULL) ); */
  print_array(arr, NUM);
  quicksort(arr, NUM);
  print_array(arr, NUM);
  return EXIT_SUCCESS;
}
```

Step by Step

Save the code to a file called easy_qsort.c. Next, compile the code:

```
$ gcc -Wall -pedantic -ansi -o qsortof -g \
easy_qsort.c
```

The most important argument to GCC is notably -g, which adds debugging symbols to the code.

Try running the program to make sure everything is kosher:

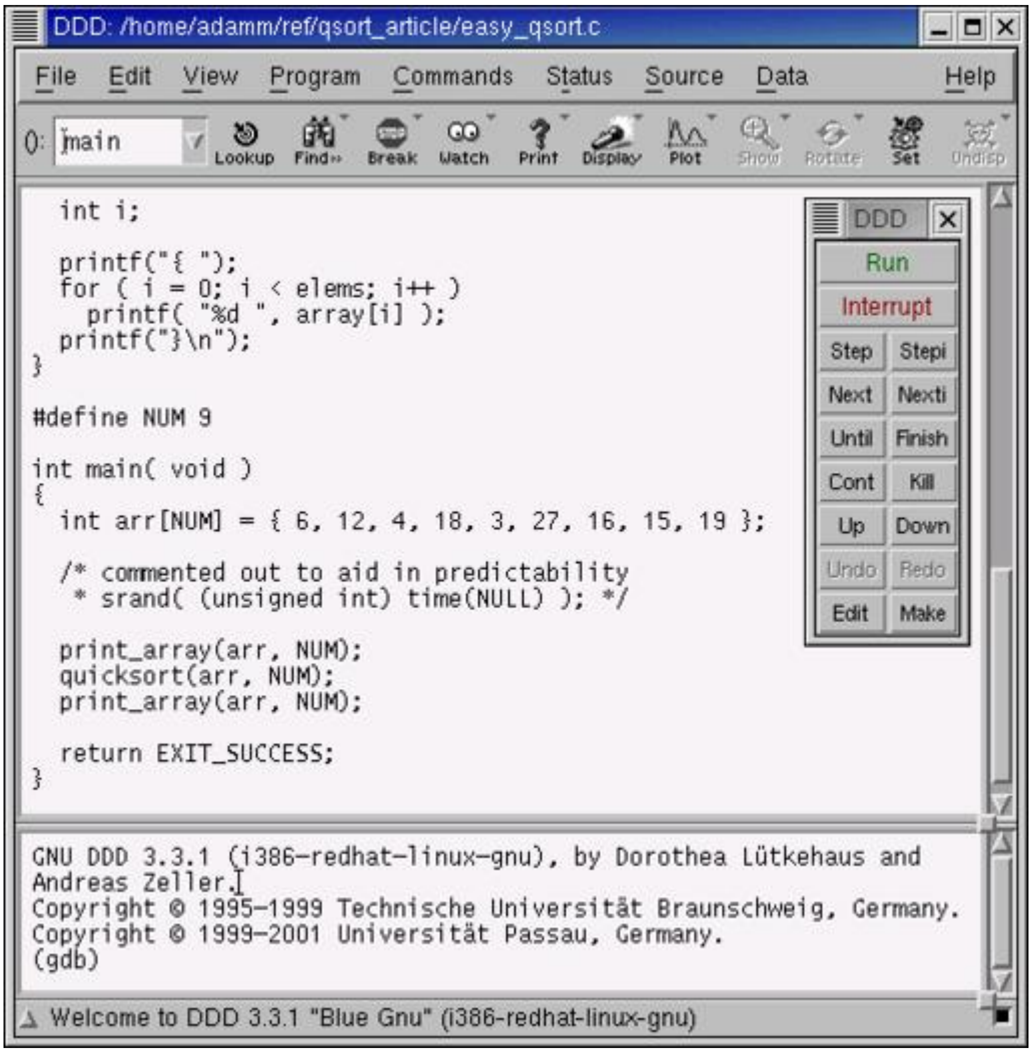
```
$ ./qsortof
{ 6 12 4 18 3 27 16 15 19 }
{ 3 4 6 12 15 16 18 19 27 }
```

The first line of output is the unsorted array, the second line is the array after running through the quicksort function.

So how does it work? Let's turn to our new friend DDD:

```
$ ddd qsortof
```

This should bring up DDD. Close any Tips and About:Help windows that pop up, and you should see something like Figure 1.



```
  int i;

  printf("{ ");
  for ( i = 0; i < elems; i++ )
    printf( "%d ", array[i] );
  printf("}\n");
}

#define NUM 9

int main( void )
{
  int arr[NUM] = { 6, 12, 4, 18, 3, 27, 16, 15, 19 };

  /* commented out to aid in predictability
   * srand( (unsigned int) time(NULL) ); */

  print_array(arr, NUM);
  quicksort(arr, NUM);
  print_array(arr, NUM);

  return EXIT_SUCCESS;
}
```

```
GNU DDD 3.3.1 (i386-redhat-linux-gnu), by Dorothea Lütkehaus and
Andreas Zeller.
Copyright © 1995-1999 Technische Universität Braunschweig, Germany.
Copyright © 1999-2001 Universität Passau, Germany.
(gdb)
```

Welcome to DDD 3.3.1 "Blue Gnu" (i386-redhat-linux-gnu)

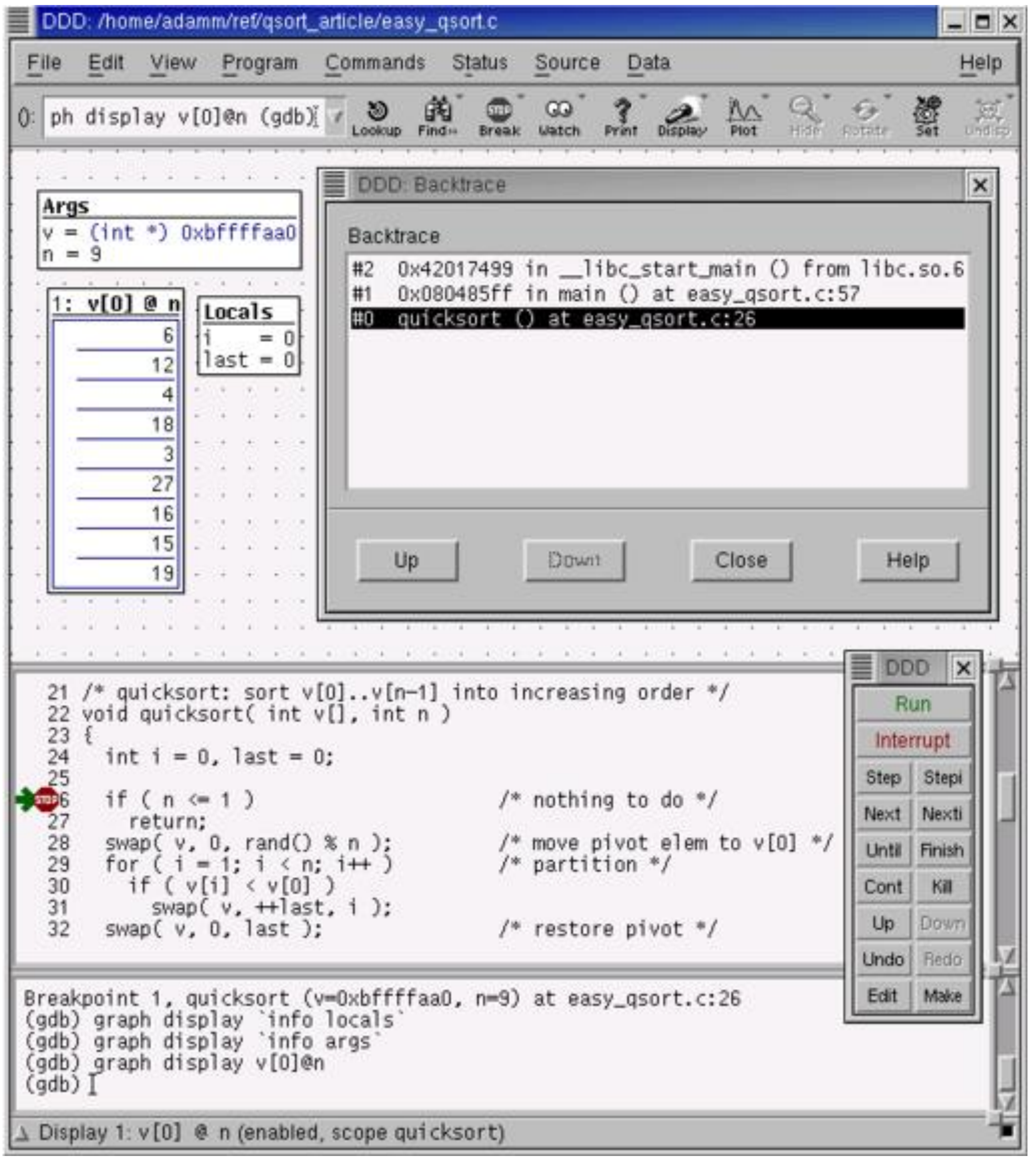
Figure 1. Freshly Started DDD

It would be a good idea to turn on line numbering now. Click the check box next to Display Source Line Numbers in the Edit-->Preferences-->Source menu. Now we can add a breakpoint and start debugging.

First, select the nothing to do line by clicking on its line number in the margin. Then, click the Set/Delete Breakpoint at () button, and click the Run button in the floating command tool.

At this point you should see a red stop sign at the line with the breakpoint and a green arrow on the same line (the code that is about to execute). Let's use DDD to display some inside information.

To begin, select Data-->Display Local Variables. Next, select Data-->Display Arguments, and then select Status-->Backtrace. Finally, type **graph display v[0]@n** into the console window, and press Enter. This displays elements 0 through n of the v[] array (see Figure 2).



```
Args
v = (int *) 0xbffffaa0
n = 9
```

```
1: v[0] @ n
6
12
4
18
3
27
16
15
19
```

```
Locals
i    = 0
last = 0
```

```
Backtrace
#2  0x42017499 in __libc_start_main () from libc.so.6
#1  0x080485ff in main () at easy_qsort.c:57
#0  quicksort () at easy_qsort.c:26
```

```
21 /* quicksort: sort v[0]..v[n-1] into increasing order */
22 void quicksort( int v[], int n )
23 {
24   int i = 0, last = 0;
25
26   if ( n <= 1 )                       /* nothing to do */
27     return;
28   swap( v, 0, rand() % n );           /* move pivot elem to v[0] */
29   for ( i = 1; i < n; i++ )           /* partition */
30     if ( v[i] < v[0] )
31       swap( v, ++last, i );
32   swap( v, 0, last );                 /* restore pivot */
```

```
Breakpoint 1, quicksort (v=0xbffffaa0, n=9) at easy_qsort.c:26
(gdb) graph display `info locals`
(gdb) graph display `info args`
(gdb) graph display v[0]@n
(gdb)
```

```
Display 1: v[0] @ n (enabled, scope quicksort)
```

Figure 2. Break at if (n <= 1)

Watch the Engine Crank

The breakpoint is set so that we don't do anything if the array we were given contains one or less elements. Because this is a recursive quicksort, this test is necessary to end recursion when an array with one element is passed in (more on this later).

After this test, we pick our pivot and move it to the beginning of the array. Click the Next button until the green arrow points to the call to the swap function. Next continues to the next line without descending into subroutine calls. Step would attempt to step into other subroutines. Now, click Next once more to see what happens.

My machine swapped the 0th and 1st elements of v[], meaning rand() % n returned 1. If you debug this program a few times you may notice that rand() % n always returns 1. Not very random, you say? In this example, by commenting out the call to srand(), the pseudo-random generator is never seeded and rand() returns predictable results.

The chosen pivot will serve to divide numbers smaller than itself from numbers larger than itself. The pivot was moved to v[0] because we don't know where it should be until we examine the entire array.

The “partition” loop steps through each element of the array and compares it to the pivot (the number 12, in my case). Last is pre-incremented, so the element at index 1 is swapped with itself (I know it's a waste). Optimizing this algorithm is left as an exercise for the masochistic reader. Note that you can click Interrupt, then Run to start over at any time.

Click Next until i equals 3 and last equals 2 (watch the Locals display in the graphical Data Window). The “if” test inside the partition loop is now comparing 18 to 12. The test fails (Next), so i is incremented (Next), and last still equals 2.

Keep Next-ing until i equals 9. My array is now { 12 6 4 3 18 27 16 15 19 }. Another click on Next and 3 is swapped with 12, seated between the smaller numbers from the larger ones.

After restoring the pivot value to its original place, we recurse into quicksort, sending a pointer to v[0] and telling it to expect a three-element array (the smaller numbers). Then we recurse into quicksort again, sending a pointer to v[4] and announcing a five-element array (the larger numbers). Those quicksort calls again recurse until only one-element arrays are passed into the recursive calls. Only then will the recursive calls return—deepest call first—until we return to the main function with a sorted array. Whew!

Handy Features

If you get deep into recursive quicksort() calls, you'll notice that the v[0]@n display is disabled. Adding a button makes re-creating this display a snap. To create the button, click Commands-->Edit Buttons...-->Data Buttons. Into the text entry field, enter:

```
   graph display v[0]@n // Varray
```

A button titled Varray should pop up at the top of the Data Window. When the v[0]@n display reads (Disabled), right-click on the display and select Undisplay. Then, click the new Varray button. If it is still disabled, try Next-ing a few times and clicking the button again.

Stack Backtrace

Previously, I had you turn on the stack backtrace window. It's interesting when you're deep in quicksort() calls to examine the stack by clicking on other lines in the backtrace window. You can see how the current calling context was reached and what the data looked like at different points in the stack.

Machine Code Window

If you're really sick and twisted, try View-->Machine Code Window to see the actual assembler instructions being executed.

Other Algorithms/Data Structures to Try

DDD is great at displaying linked lists and other data structures. Try it! Also, I've mentioned that this quicksort implementation is nowhere near optimized. To see a highly optimized version, check out qsort.c in the GNU C library.

Resources

email: adamm@wazamatta.com

Adam Monsen is a recovering Pre-Med student, now a “software engineer” at a Seattle, Washington-based startup called Classmates.com. His hobbies include playing the piano, surfing and cat juggling. Adam likes coding Perl, Java and sometimes even C on his Red Hat GNU/Linux 7.3 desktop.

Power Sessions with Screen

Adam Lazur

Issue #105, January 2003

Adam explains the many benefits and uses of screen.

Screen is a terminal multiplexer that allows you to manage many processes through one physical terminal. Each process gets its own virtual window, and you can bounce between virtual windows interacting with each process. The processes managed by screen continue to run when their window is not active.

Thus far, the screen features described aren't all that exciting or new. In fact, there already are X11 terminal applications that provide this functionality (konsole and multi-gnome-terminal). What differentiates screen from the others are some of the core features screen provides.

Screen offers the ability to detach from a session and then attach to it at a later time. When detached from a session, the processes screen is managing continue to run. You can then re-attach to the session at a later time, and your terminals are still there, the way you left them.

Screen also maintains individual, searchable scrollback buffers for each of the windows it manages. You can perform traditional “enter the search term and I'll find it for you” searches as well as incremental searches. This is such an obvious feature, it's surprising that more terminal emulators do not offer it.

Other notable features of screen are configurable key bindings, utf8 and multibyte charset support, multi-attach, configurable input and output translation, input and output filter, multi-user support with access control lists (ACLs) and logging.

Interacting with Screen

Before actually running screen, it's important to understand how to interact with it. Screen sends all entered text to the current window, with the exception of the command character. The default command character is Ctrl-A (press the Ctrl and the A key at the same time). The screen man page uses C-, Emacs style, to mean Ctrl-.

The command character is used to notify screen that you'd like to control screen itself, rather than the application in the current window. The key pressed after the command character designates which screen command you would like to perform.

Some of the more useful commands and their key bindings are shown in Table 1.

<u>Table 1. Commands and Their Key Bindings</u>

For many of the commonly used commands, the control version of the key is also bound to the command. An example of this is Ctrl-A C and Ctrl-A Ctrl-C to create a window.

To send Ctrl-A to an application without screen intercepting it, you can press Ctrl-A A. The command character can be changed to an alternate key if you wish. Typically Emacs users change the command character to Ctrl-B by adding **escape Bb** to their .screenrc. The following examples use Ctrl-A because that is the default.

As you might expect, .screenrc is the per-user configuration file in your home directory, and /etc/screenrc is the system-wide configuration file that applies to all users.

Window Basics

Now that you understand the fundamentals of interacting with screen, we can step through screen fundamentals using a typical screen session as an example. Probably the most typical use of screen is to control terminals on a remote machine on which you have a shell login.

So, for those of you playing along at home, log in to a remote host that has screen installed. If you don't have a remote host to **ssh** to, you can install screen and **ssh** to localhost. Packages for screen are available for most distributions.

You're now sitting at the shell prompt on the remote machine. Type **screen** at the prompt. You should see a splash screen showing some information explaining that screen is under the GPL, where to report bugs and so on. You can press the spacebar to bypass this screen (you can disable this splash screen permanently with **startup_message off** in your .screenrc). Next, you should arrive at another shell prompt, this one running inside of screen.

The new shell running inside of screen should behave like your first shell would. If you do a **printenv**, you may notice a few new environment variables set. Screen sets TERM to screen—each screen window provides its own vt100-compatible virtual terminal. The variable WINDOW is set to the virtual window number, and the variable STY is set to your session name. I'll explain more about those last two later.

So far, this single shell works exactly like you're used to working on a remote machine. For the sake of this example, say we are downloading the current version of screen (as of this writing, screen 3.9.13) from the screen distribution site at ftp.uni-erlangen.de/pub/utilities/screen. While this file is downloading, you decide to use your spare time to clean up your home directory. If you weren't using screen, you'd have to open another xterm and **ssh** to the remote machine. With screen, a simple Ctrl-A C will create a new screen window with a new shell process.

Thus far, you have an FTP client and a shell busily tidying up your home directory. You can check on your download in the original FTP window by using Ctrl-A P to go to the previous window. You can get back to your shell with Ctrl-A N.

When you check on your download, it's not finished (screen doesn't take that long to download, but we'll pretend it does for this demonstration). Time to get back to your messy home directory, right? Before doing that, press Ctrl-A Shift-M to monitor the current window for output. Now screen will notify you when there is activity in the FTP window. Bouncing between windows to check on your download's progress is no longer necessary. This also works in the inverse case; use Ctrl-A _ to monitor for silence (30 seconds by default). Monitoring for silence is useful for long compile jobs or other things that spew information.

You can continue to spawn new shells and do things in parallel on the remote machine. After you open a few windows, it becomes difficult to keep track of which window is where. This is where the windowlist comes in to play. Press Ctrl-A “, and you will be presented with a list of the current open windows. Navigate the list with the J and K keys. Pressing Enter on an entry will make it the current window. By default, the window name isn't all that descriptive. You can remedy this by setting the window name yourself with Ctrl-A Shift-A. You can set these titles automatically, much as you would set titles in an xterm, by sending Esc-K, then the title, then Esc-\. Most likely you can adapt a shell-specific recipe for setting the xterm titlebar to the screen window name using the mentioned escape sequences.

To finish up our basic run-through of windowing, let's close your existing windows. If you exit the shell that screen has spawned, the window is deleted automatically. You can delete a window manually with the kill command (default Ctrl-A K). When you exit all of the screen windows, screen exits. You also can tell screen to exit and kill all of your windows by issuing the quit command (Ctrl-A \).

Sessions

Now that you're creating new windows and bouncing around between them, you can get many things going in parallel. You could potentially have some editors, an IRC client and a few other things all running in their own windows. But occasionally disaster strikes, and your network connection dies (those of you still playing along at home can kill your SSH client). It looks like it's time to pick up the pieces and relaunch all of your applications on the remote machine, right? Not with screen.

Each time you start up screen without arguments, it creates a new session. This spawns two processes: a terminal management process and a client process. The client process automatically is “attached” to the terminal management process. When you type, the characters you enter go to the client, which sends them to the terminal management process, which then sends them to your application.

When your network connection dies, the client catches the signal and detaches from the terminal management process. The terminal management process continues along managing your terminals as if nothing happened. When you log back in, you can list running sessions by issuing **screen -ls** at the prompt. It should show something similar to the following:

```
There are screens on:
        24319.pts-9.hostname (Detached)
1 Sockets in /var/run/screen/S-youruserid.
```

This shows that your session automatically detached when your connection dropped.

You can re-attach to the session in a few ways. You can give a session name explicitly with **screen -r *sessionname***. You can tell it to re-attach if possible, otherwise start a new session by running **screen -R**. Or you can go the “do whatever is needed to get a screen session” route and run **screen -D -RR**. This last option will detach already-attached clients and attach to the first session listed.

When you run one of these commands, you should be right where you left off before your network connection went down. When you're re-attached, you can continue working as if nothing ever happened.

It is also possible to attach to a session multiple times. This is useful if you haven't closed your screen session from another machine, or if you simply want to display windows from the same session side by side. You can multi-attach by adding an **-x** in the command-line options to screen when attaching.

Finally, when the end of the day rolls around and it's time to go home, you can detach from your session using Ctrl-A D. When you return the next day, you can re-attach, and you will be back where you left off.

Copy and Paste/Scrollback Mode

One of the key features listed in the beginning of the article was screen's searchable scrollback. This is a feature I could not live without. It's not immediately obvious to the new screen user, but screen's scrollback is accessed via the copy command. (You can enter copy mode with Ctrl-A [or via the **copy** command.) Navigation works as expected with either the Arrow keys and Page Up/Down or the vi motion equivalents. Searching is accessed via either / and ? for vi-style search or Ctrl-S and Ctrl-R for incremental search. Case-insensitive search can be turned on with the screen command **ignorecase yes**. If you are using copy mode for scrollback only, it can be exited at any time with the Esc key.

To copy text, maneuver the cursor to the beginning of the desired text, and press the spacebar to mark it. Then position the cursor over the end of the text you'd like and press the spacebar again to mark it. When you mark the end, the text is copied into screen's internal copy buffer, and copy mode is exited. You can paste the text in your copy buffer into the active window with Ctrl-A].

The final thing you should know about the copy and paste mode is the scrollback buffer is limited to 100 lines by default. This is, in my opinion, not enough. You can tweak this to a higher value (1,024 for example) by adding the command **defscrollback 1024** to your .screenrc.

Opening Windows with screenrc

I have already mentioned that you can add a command to your .screenrc to change the behavior of screen. It's not immediately obvious, but you can put any screen command in a screenrc. This is very useful and can be used to spawn windows automatically with the **screen** command.

A typical application of this tidbit of knowledge is to launch a predefined set of windows at screen startup. Below is a sample screenrc that will do so:

```
# read in your normal screenrc
# before anything else
source $HOME/.screenrc
# now start opening windows
screen top
# it's possible to set the window title with
# the -t option
screen -t irc epic
# you can also specify the window number
# to launch in
screen -t mail 8 mutt
screen -t daemon 9 tail -f /var/log/daemon.log
```

If you save this to $HOME/.screenrc.multiwin you can tell screen to use it instead of your normal .screenrc by running **screen -c $HOME/.screenrc.multiwin**.

You also can launch more systems-oriented screen sessions from a startup script. A common application of a system screen session is a serial console server. Screen is well suited for this task because it has built-in support for serial terminals and logging. A commented example of a screenrc for this purpose is:

```
# This assumes that serialuser has proper
# permissions to access the serial ports and to
# write to the log files specified in the screenrc.
# turn logging on for all windows
deflog on
# tell screen to log to /var/log/serial.$WINDOW
logfile /var/log/serial.%n
# open windows on the serial ports
screen /dev/ttyS0 38400
screen /dev/ttyS1 19200
```

If you saved this file in /etc/screenrc.serial, you could launch it at startup with a script that runs:

```
su serialuser -c \
'screen -dmS serial -c /etc/screenrc.serial'
```

The **-dmS serial** options tell screen to launch the session in detached mode and name the session “serial”. User serialuser can log in and attach to this session exactly like any other normal screen session. Launching a detached screen also can be used to start screen from a cron job if this is preferred.

It is possible to set up a single system-wide screenrc that allows multiple users to connect to it. Screen supports multi-user mode with per-window ACLs that define what each user can and cannot do. Multi-user screen sessions, however, require that screen be setuid root. Because of this requirement, I am not going to include examples for multi-user screen sessions in an introductory article. If you would like to set up a multi-user screen session, read the screen docs, put on your “adding setuid root permissions to a complex piece of code” paranoia hat and be prepared to lock things down as tightly as possible.

As a third application, you could merge the two previous examples and launch system-wide interactive programs via screenrc. A good use of this would be launching mutella, a curses-based gnutella client, at startup. With screen, you can launch this program and connect to it on occasion to see the status, run queries, etc.

Further Information

You can find further information on screen in the screen documentation. The documentation is provided in both man and info format. I prefer the info format when browsing and the man page when searching for specific things, but that's a personal preference.

There are also a few on-line resources for screen users. First is Sven Guckes' screen pages at www.math.fu-berlin.de/~guckes/screen. Second is the helpful screen mailing list at groups.yahoo.com/group/gnu-screen. The mailing list is the first place you should go with questions after you have exhausted the available documentation. You must be subscribed to post.

Adam Lazur is a Linux consultant doing things ranging from embedded Linux to Beowulf clusters. In his spare time, Adam likes to write about himself in the third person. Adam welcomes comments about this article at adam@lazur.org.

Archive Index Issue Table of Contents

Advanced search

Must-Have Zaurus Hardware and Software

Guylhem Aznar

Issue #105, January 2003

Tips on ways to optimize your Zaurus and some killer applications.

You just got your Zaurus; you tried every single application and even typed some commands in the terminal to prove it really is a GNU/Linux machine in the palm of your hand. You know where most of the keys are but still may be looking for the pipe (hint: it's Shift-Space). So now what to do? This article presents some recommended upgrades and killer applications.

Expanding Memory

You will need more than the 32 or 64MB of RAM that comes with the Zaurus, because the system uses most of that memory even if you haven't installed anything. With a stock 32MB 5000d using ROM 2.37, only 600k of memory are free without any application installed or started. You have two places to put more memory: the SD/MMC slot or the CompactFlash slot. You need the CF slot for a network card, but SD/MMC can be tricky.

SanDisk cards, the most popular and cheapest SD cards, are prone to failure. Sharp even had to release a new driver to fix some problems with the less than $50 US 128MB SanDisk cards that would fail after repartitioning or work only once. The real problem is SD drivers can never be free software because of the copy-restriction system. Get an MMC card instead; free drivers are available.

Case and Accessories

The Zaurus is a hot seller in Japan, and diverse accessories are available there. Do yourself a favor and at least get a leather case from Extreme Limit. The Portfolio model will protect your Zaurus perfectly. Another good case is the GLP-824 iPAQ case from Sumdex, which can be zipped for full protection.

I suggest a second battery and charger as well. If you keep your Zaurus in the case instead of docking it, you also will need a USB charging sync cable. However, don't get a serial cable; it is badly designed and will prevent you from opening the keyboard.

Updating the SL-5000D…and the 5500

The 5500 has very few differences from last year's developer model, the SL-5000D. The 5500 comes with Hancom Office, a better metal and plastic stylus and a better AC adapter. Although the Zaurus doesn't have a built-in microphone, the die-hard hardware modification crew can build one in (see Resources).

ROM: Sharp, Opie, Paul or Crow?

Now that your hardware is ready, it's time for some software tweaks. Many different ROM images, or ROMs for short, are available (see Resources). Creating your ROM also is possible with modzaurus. If you have the Sharp ROM version 2.37 or earlier, there is a known security issue, so you should upgrade.

Which ROM is best? Sharp is the default ROM sold with the Zaurus and is widely supported. Opie is a pure, free software ROM with improved default applications. Paul ROM is the stock Zaurus ROM with some modifications to use the MMC card as the storage media, keeping the whole memory available as RAM. Crow ROM is the equivalent of Paul ROM with Opie software.

Opie's overall design is better, with a filesystem more like standard GNU/Linux, and it includes more recent versions of applications than Sharp ones. However, it lacks the Java VM and the Opera browser, and its improved filesystem makes it incompatible with some third-party software. If you have an MMC card, you really should consider Paul's or Crow's. Simply format your MMC card as ext2, and it will be mounted under /home. Opie also can use an MMC card but has compatibility problems.

The best ROM would include improved Opie applications, let you install non-free software (at least Opera, Jeode, Hancom Office and theKompany.com applications) and work flawlessly. Unfortunately, it does not exist—yet.

The ROM updating process requires care. First, download the ROM version you need. Remember, the latest is not always the best. Once you have downloaded the ROM file of your choice, put in a CF card using a USB CompactFlash reader or a PCMCIA converter for CompactFlash cards. First, check that your CF card is formatted as FAT16, then rename your chosen ROM file to Romimage, and put it in the root directory of the CF card. Check that the entire file copied correctly to CF by comparing the file length and md5sum to the original.

Next, if you have any important data in your Zaurus, back it up. Any data will be lost during the ROM update process. Now, turn off your Zaurus, and plug in the AC power. The orange charging-battery LED will turn on. Then, open the battery compartment lid, but do not take the battery out. Now the tricky part: while pressing the C and D keys on the keyboard at the same time, press the full Reset button once. This button is located below the battery compartment and can be accessed with a stylus.

You may need a friend to help with this awkward key-pressing sequence. If you are successful, both LEDs will turn on and the update process will start. Don't touch anything—an incomplete ROM update may damage the Zaurus, so have a coffee and come back three minutes later. You will find both LEDs turned off. Then, you can eject the CF card, close the battery compartment and press the Reset button.

The most common way for this to fail is if you put an incomplete Romimage file on the CompactFlash. Check after copying and before inserting the CF card in the Zaurus.

PIM Applications

Migrating data from your old PDA, as explained in the Zaurus FAQ, is troublesome. Beaming database after database, then bits and parts of other databases, is tiring. But, now there is an easy way to migrate your PIM through a simple Perl script. Install Perl 5.6.1, opie-sh and p2z. Next, put AdressDB.pdb, DatebookDB.pdb, MemoDB.pdb and ToDoDB.pdb on your Zaurus, and then run **p2z**.

Now for the cell phone. If your cell phone has an IR port that can send data, receive it on the Zaurus by clicking Settings-->Beam receive. Click on Add to Address book or Add to Calendar when prompted to include the received information in the relevant applications.

The clunky bundled personal information manager (PIM) applications are the Zaurus' big weak point. An alternative is tkcAgenda, tkcDatebook and tkcMemo from theKompany.com. While theKompany.com's PIM applications are fully compatible with the default applications, they replace them completely without leaving you any choice or restoration option.

theKompany.com also offers tkcKapital to help you manage your money and tkcExpense, a nice add-on to track your expenses.

Sync (with Free Bonus Networking)

Unlike older PDAs, the Zaurus actually talks IP over the USB cable to the host computer. If you're running masquerading on the host, the same connection you use for synchronizing data becomes a general-purpose net connection. Instructions for setting up networking over USB for several common distributions are available (see Resources). After you have that working, use the Qtopia Desktop software from Sharp or direct from Trolltech to sync.

Getting on the Internet

With USB, you have a working net connection when the Zaurus is in the cradle. If you only have a wired network, you can get a CompactFlash network card for around $50 US; however, wireless networking is much more fun. The best 802.11b card available for the Zaurus is the Socket WL6000-320 low-power card. While bigger cards like the Linksys WCF-11 work, they eat a lot of power. The Linksys WCF-11 also prevents you from using the stylus slot. Some other cards, including D-Links, block both the audio port and the stylus slot.

Some configuration file tweaks are necessary if you have a ROM prior to 1.1x. But because early Sharp ROMs have a security hole, it's best to upgrade your ROM before adding a network card.

The Socket WL6000-320 also will certainly become plug-and-play in the future—maybe by the time you read this. Check your ROM's release notes, and if support hasn't been added, use the driver listed in Resources.

Now that you have an 802.11b-capable Zaurus, you might want an access point. I recommend the hacker-friendly Linksys WAP11. Not only does it work well with GNU/Linux using SNMP software, but you can update its firmware and use directional antennas to make a long-distance wireless bridge.

Installing Applications with ipkg

The problem of packaging applications for Linux PDAs was solved when the first versions of Linux running on the Compaq iPAQ appeared. An ipkg package is basically a .tar.gz file with some basic control information. If you want to install an application, type:

```
ipkg install package.ipk.
```

Make sure it was compiled for ARM and that you have the libraries it will need. If you want to remove a package, **ipkg remove *package*** will do it.

Now, the Essentials…Games!

The best way to play a lot of games on the Zaurus is to install an emulator. What about playing old Nintendo Game Boy games? If you have purchased the games, get GnuBoy, and find a way to download the ROM files legally for the game you own. You can get a GnuBoy application associated with all of the *.gb and *.gbc files, so you will be able to start each Game Boy game simply by selecting the files from the Documents tab. Add the following line to /home/QtPalmtop/etc/mime.types:

```
application/gameboy gb gbc
```

And add these lines to qtopiagnuboy.desktop:

```
MimeType=application/gameboy
        MimeTypeIcons=GnuBoy
```

snes9x is the famous Nintendo emulator, which works much the same way for SuperNES games. However, native games will run better and faster than either Game Boy or SuperNes emulated games.

You must know *NetHack*--wouldn't you like to play it on your Zaurus as well? If you prefer a chess-like strategy game, the Zaurus is a Java-capable device, so you can get *Laser Chess*. And don't miss *Froot*, a puzzle bubble-like game that won the Games category in the recent Qtopia developer contest (Figure 1).

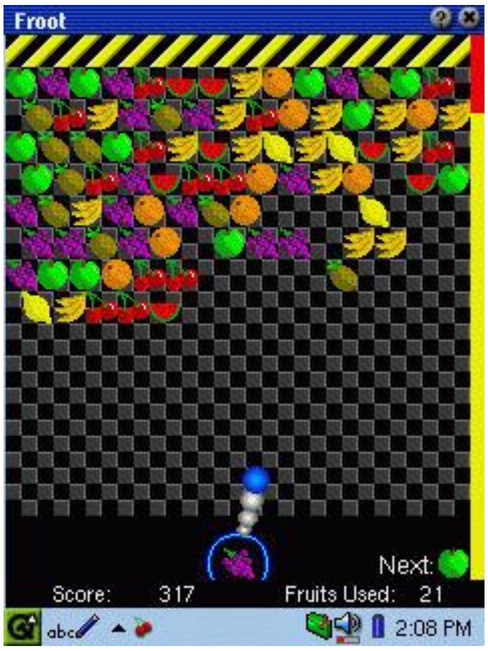


Figure 1. The award-winning *Froot* is simple, yet addictive.

MPEG and Divx Movies

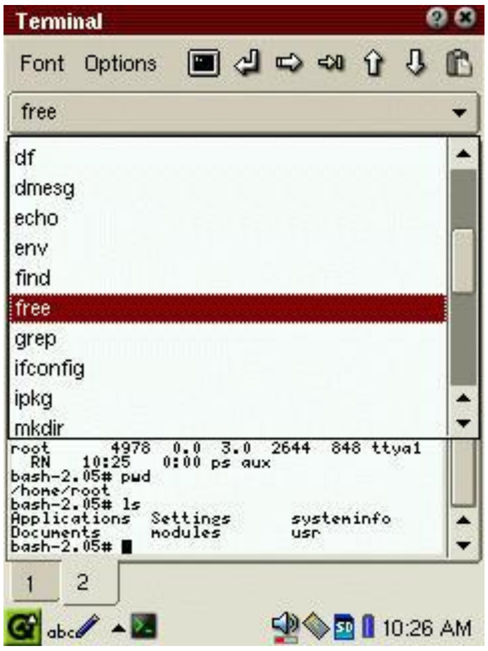
Go to www.pocketmovies.net and download a trailer. Then start Applications-->Media Player, choose Options-->Full screen, plug in some earphones and enjoy! The quality is excellent for such a small device. You can play movies in MPEG-1 SIF (320 × 240) format at 24 frames per second. If you want more frames per second or want to store more movies, HALF SIF (160 × 120) is also possible. The Fit to Screen option will make the low resolution nearly unnoticeable.

More full-featured media players are available. The opie-mediaplayer2 is a media player that supports MP3, Shoutcast, Ogg Vorbis, Divx and others, but it uses a lot of memory. A better performing player for Divx movies is tkcVideo, which will drop frames to keep a smooth playback and a good synchronization.

Utilities

Tab Manager lets you easily reorganize applications in the launcher. If you are going to run background processes, get Process Manager. The kill and renice features are very useful for dæmons, console applications or just killing unstable processes.

Next, let's add SSH and a better terminal. The KDE konsole terminal, embeddedkonsole-tabs, is ported to Qtopia and available on SourceForge (Figure 2). Now, you can use different tabs if you need more than one terminal open at once. While you're at SourceForge, pick up SSH. Your home directory is not writable, so SSH can't run. It needs to write to .ssh in your home directory. Edit /etc/passwd, and set /home/root as the home directory instead of /root. You also can get a Zaurus version of Emacs from this site.



```
root        4978  0.0  3.0  2644  848 ttya1
  RN    10:25   0:00 ps aux
bash-2.05# pwd
/home/root
bash-2.05# ls
Applications  Settings       systeminfo
Documents     modules        usr
bash-2.05# 
```

Figure 2. embeddedkonsole-tabs is a KDE-like tabbed terminal.

You should have a password set to protect your Zaurus if you hang around with a 802.11b and OpenSSH accepting root connections without any password. Use Settings-->Security to set a password.

To use your Zaurus remotely from anywhere on the network, install fbvncserver. Then, somewhere on your network, start a VNC client; the Zaurus display is exported.

If you want to take a screenshot of the applications you are evaluating, using the screenshot applet is straightforward. It should be run in the terminal with:

```
delay: sleep 2 ; scrshotcf
```

Various password managers that use encryption to protect your passwords are available on the Zaurus. Keyring has a user-friendly interface and uses blowfish.

If you already have a lot of information on STRIP for PalmOS, you can convert your password file to ZSafe format using strip2zsafe. ZSafe, although not as user-friendly as Keyring, is very easy to migrate to through its plain-text import option. Create a text file using the following format:

```
"Category";"Name";"Username";"Password";"Comment"
```

and put it on the Zaurus. Then, select the text from ZSafe, and it will be added to your password file and encrypted for the next time you use it.

Maps, Drawing and Notes

In case you've plowed through the tools and utilities already presented, here are some more examples of the flurry of activity surrounding the Zaurus. Get qpeGPS to read mapblast.com maps on your Zaurus and automatically follow your road.

Various picture viewers are available. Consider tkcGallery from theKompany.com and its free software equivalent, MooView.

drawpad and IQNotes (iqnotes.kybu.sk) are daily companions. drawpad (Figure 3) lets you draw something on your Zaurus screen when you can't type, and IQNotes (Figure 4) keeps text notes and sketches in a convenient hierarchical tree.

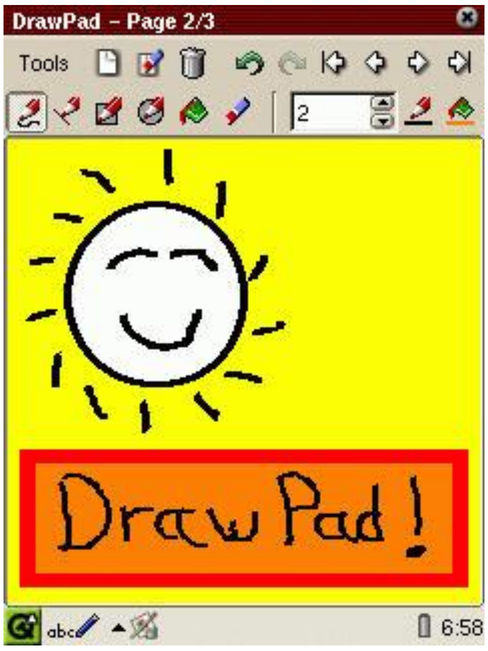


Figure 3. drawpad—does what it sounds like.

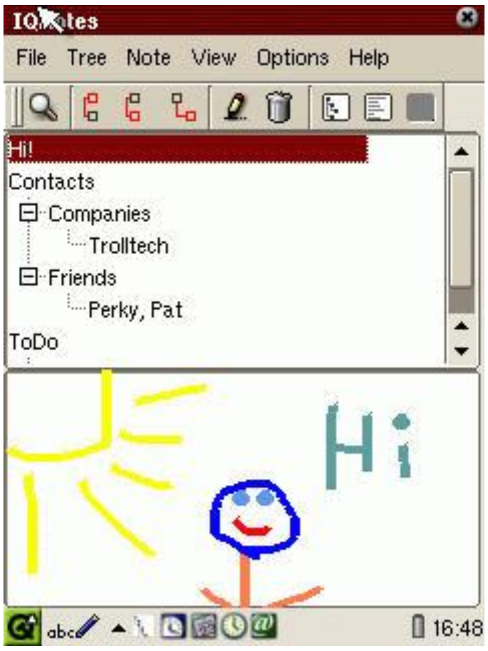


Figure 4. IQNotes keeps notes and drawings organized.

A shopping list is the killer application for getting your PDA into the family. Shopper will make your Zaurus more acceptable, turning it from a geek toy into a useful companion. Add shopping list items by store, and show your list for each store as you go.

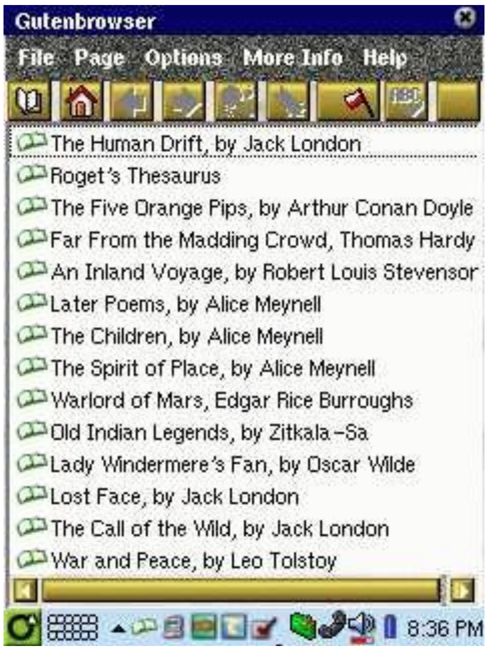
Exploring the Sky

Don't let your lack of star knowledge ruin your romantic dates in the moonlight. Download and install the rather large ZaurusSkyExplorer, and its 4.5MB of knowledge will help you with constellations and star names. If you need to go further and see where satellites are, PetitTrack and TLE files from NASA will help you.

Voice and eBooks

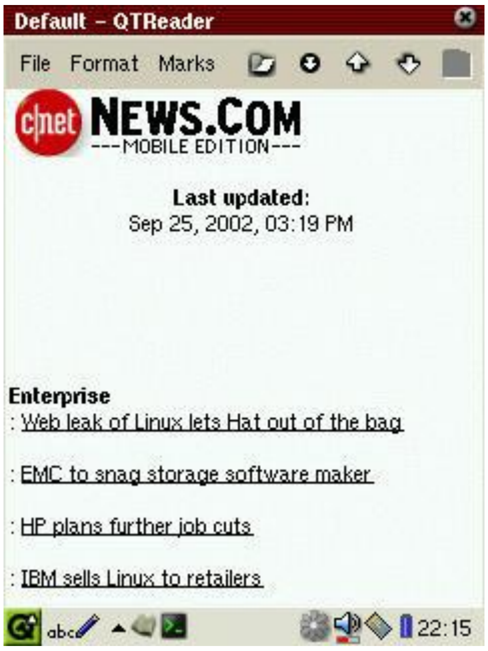
The excellent CMU speech tools have been ported to the Zaurus. They can read any text file aloud. The British voice has a heavy accent, but its uniqueness makes it a must-have to impress other PDA owners or to quietly read today's news headlines. We can only hope it will be integrated tightly into eBook applications.

Gutenbrowser searches, downloads and lets you read free classic literature from Project Gutenberg in the palm of your hand (Figure 5). A nice complement is QT Reader, which reads many different documentation formats, from text files to palm .pdb documents or plucker documents (Figure 6). With the latter feature, you can fetch a batch of documents to your Zaurus and read them off-line. It is up to you to decide which news sources, comics or web pages you want to include. You even can create your own starting page.



The Human Drift, by Jack London
Roget's Thesaurus
The Five Orange Pips, by Arthur Conan Doyle
Far From the Madding Crowd, Thomas Hardy
An Inland Voyage, by Robert Louis Stevensor
Later Poems, by Alice Meynell
The Children, by Alice Meynell
The Spirit of Place, by Alice Meynell
Warlord of Mars, Edgar Rice Burroughs
Old Indian Legends, by Zitkala-Sa
Lady Windermere's Fan, by Oscar Wilde
Lost Face, by Jack London
The Call of the Wild, by Jack London
War and Peace, by Leo Tolstoy

Figure 5. The public domain in your PDA: Gutenbrowser.



NEWS.COM
---MOBILE EDITION---

Last updated:
Sep 25, 2002, 03:19 PM

Enterprise
: Web leak of Linux lets Hat out of the bag
: EMC to snag storage software maker
: HP plans further job cuts
: IBM sells Linux to retailers

Figure 6. Grab now, read later with plucker and QT Reader.

plucker is 100% compatible with Avantgo if you know the URL of the content. The hardest part is finding PDA-optimized content, because the Avantgo license prevents web sites from publicly disclosing the PDA-optimized URL or serving it to other off-line readers.

However, with some time you can find newspapers such as *The Age* from Melbourne, Australia that give a link to their handheld version somewhere on their first page. Look at the tiny print.

Collect these URLs or design your own page and launch Spider.py, a plucker Python browser. It will create a .pdb file that can be read on the palm at any time.

Math

Although the Zaurus comes with a good calculator, it does not let you convert units easily because of the lack of programmable functions. ZUC is an excellent converter. Two featureful math tools are proprietary, NeoCal and Formulae 1. NeoCal is currently the best calculator available for the Zaurus and offers a lot of financial, scientific and statistical functions. Formulae 1 is an equation solver that displays not only the results but the intermediate steps. A math assistant worth purchasing if you need some e-help.

Files, Scripts and Directories

Before we go on with network applications, you should know where they are installed on the Sharp filesystem:

- /home/root/usr/lib/ipkg: a list of the files that were installed for each application.
- /home/QtPalmtop/apps: description files for the launcher.
- /home/QtPalmtop/bin: executables.
- /home/QtPalmtop/pics: icon files in PNG format and other pictures.
- /home/root/Applications: data files for applications (most of which are stored in XML).
- /home/root/Settings: configuration files for some of the applications.

If applications are installed onto a memory card, a QtPalmtop directory is created on the card. You also can tweak application appearance that way.

Code to be executed at startup should be installed in a file in /etc/rc.d/rc5.d. A good choice for a filename might be S99local. This example sets environment variables correctly and sets the hostname:

```
#! /bin/sh
#
# S99local - run local startup stuff
#
PATH=/bin:/usr/bin:/sbin:/usr/sbin
LD_LIBRARY_PATH=/lib:/usr/lib:/home/QtPalmtop/lib
export LD_LIBRARY_PATH PATH
hostname myhost.my.domain.us
```

Be sure to make S99local executable by entering:

```
chmod 750 S99local
```

Internet Applications

If you use instant messaging, tkcjabber from theKompany.com is worth every cent of the license fee. It will turn your 802.11b Zaurus into an instant messenger that is always on if you run it in the background. Using ICQ, AIM, MSN, Yahoo Messenger and IRC now will be exactly like using a cell phone. When you receive a message, the Zaurus incoming-message LED blinks, and a cell phone-like sound is played. Then you can chat in a window.

There are only two drawbacks. First, the interface is icon only. The first time you use it will be the worst: no tooltips, no text, no nothing. Fortunately, the user interface is very intuitive, and you won't be lost for long, except for the contacts. I have yet to find how to add contacts by ICQ number or nickname. The second problem with tkcjabber is IRC support. You are disconnected from

IRC every time you click an obscure icon. These bugs are very minor and did not interfere with the daily use of this great application.

Free software instant-messenging clients are also available: QTJim for Jabber, KinkattaLite for AIM and KMerlin for MSN. Zic will take care of the IRC part. There is no free ICQ or Yahoo Messenger software, so follow my advice and get tkcJabber.

Now, what about sniffing 802.11b traffic? Kismet is an excellent application for this. If you are using a prism2-based wireless card, like the Linksys WCF-11, you should not have any problem installing and using it. A version that works for the Socket card is also available from the same site as the Socket driver. The excellent kismet-qte provides a GUI interface. For network troubleshooting, the familiar project has ported nmap.

Conclusion

I hope this article has introduced you to some of the good software available for the Zaurus. The best part is most of these excellent applications are released under the GPL. If, like myself, you have previously used a Palm Pilot and always wanted some special software, you will be better off with the Zaurus.

Resources

email: g@7un.org

Guylhem Aznar is the coordinator of The LDP (www.tldp.org) and documentation coordinator for the GNU Project. Although he is a sixth-year medical student in real life, he is currently working on his PhD in Computer Science and with his little time left enjoys playing with the Zaurus. Reach him at externe.net.

IBM's Journaled Filesystem

Steve Best

David Gordon

Ibrahim Haddad

Issue #105, January 2003

To restart a telecom server quickly, you need a journaled filesystem. Here's how you can move to IBM's AIX-derived JFS.

New features are constantly being added to the Linux kernel; one of them is the support for journaling filesystems. JFS from IBM is one of the many journaling filesystems available now for Linux. This article explains JFS internals and characteristics, how to install and configure it on a Linux server and the operational experience of JFS at the Ericsson Research Lab in Montréal.

A filesystem is used to store and manage user data on disk drives. It ensures that the integrity of the data written to the disk is identical to the data that is read back. In addition to storing data in files, a filesystem also creates and manages information, such as free space and inodes, about the filesystem itself. Filesystem structures commonly are referred to as metadata, everything concerning a file except the actual data inside the file. Elements of the file, such as its physical location and size, are tracked by metadata.

Journaling vs. Non-Journaling Filesystems

A journaling filesystem provides improved structural consistency and recoverability. It also has faster restart times than a non-journaling filesystem.

Non-journaling filesystems are subject to corruption in the event of a system failure. This is because a logical file operation often takes multiple media I/Os to accomplish and may not be reflected completely on the media at any given point in time. For example, the simple task of writing data to a file can involve numerous steps:

- Allocating blocks to hold the data.
- Updating the block pointers.
- Updating the size of the file.
- Writing the actual data.

If the system is interrupted when these operations are not fully completed, the non-journaling filesystem ends up in an inconsistent state. In this case, these filesystems rely on their fsck utility to examine all of the filesystem's metadata (for example, directories and disk addressing structures) to detect and repair structural integrity problems before restarting. **fsck** can be rather time consuming, with the amount of time being dependent on the size of the partition, the number of directories and the number of files in each directory. In the case of a large filesystem, journaling becomes crucial. A journaling filesystem, on the other hand, can restart in less than a second.

JFS Introduction

JFS was designed to support fast recovery from system outages, large files and partitions and a large number of directories and files. To meet these requirements, JFS provides a sub-second filesystem recovery time, achieved by journaling only the metadata. JFS also provides 64-bit scalability, with petabyte ranges for files and partitions. In addition, B+tree indices are used on all filesystem on-disk structures. For better performance, B+trees are used extensively in place of traditional linear filesystem structures.

Filesystems

A file is allocated in sequences of extents. An extent is a sequence of contiguous aggregate blocks allocated to a JFS object as a unit. An extent is contained wholly within a single aggregate (and therefore, a single partition). Large extents, however, may span multiple allocation groups. An extent can range in size from 1 to 224 - 1 blocks. JFS, for example, uses a 24-bit value for the length of an extent. The maximum extent, if the block size is 4K, would be 4K * 224 - 1 bytes and is equal to (~64G). Note that this limit applies only to a single extent; in no way does it limit the overall file size. Extents are indexed in a B+tree for better performance in inserting new extents, locating particular extents and so forth.

In general, the allocation policy for JFS tries to maximize contiguous allocation by allocating a minimum number of extents, with each extent as large and contiguous as possible. This allows for larger I/O transfers, resulting in improved performance.

History of JFS

IBM introduced its UNIX filesystem as the Journaled Filesystem (JFS) with the initial release of AIX Version 3.1. This filesystem is now called JFS1 on AIX. It has been the premier filesystem for AIX for the past ten years and has been installed in millions of customers' AIX systems. In 1995, work began to make the filesystem more scalable and to support machines with more than one processor. Another goal was to have a more portable filesystem capable of running on multiple operating systems.

Historically, the JFS1 filesystem is closely tied to the memory manager of AIX. This design is typical of a closed-source operating system or a filesystem supporting only one operating system.

The new Journaled Filesystem, on which the Linux port was based, was first shipped as part of the OS/2 Warp Server for eBusiness in April 1999, after several years of designing, coding and testing. It also shipped with OS/2 Warp Client in October 2000. Parallel to this effort, some of the JFS development team returned to the AIX Operating System Development Group in 1997 and started to move this new JFS source base to the AIX operating system. In May 2001, a second journaled filesystem, Enhanced Journaled Filesystem (JFS2), was made available for AIX 5L. Meanwhile, in December 1999, a snapshot of the OS/2 JFS source was taken, and work was begun to port JFS to Linux.

Experience as an Open-Source Project

In December 1999, three potential journaling filesystems were begun or were in the process of being developed or ported to Linux. Ext2 was adding journaling to its filesystem under the name of ext3. SGI began to port their XFS filesystem from IRIX. The third filesystem was being developed by Hans Reiser and came to be called ReiserFS. But none of these filesystems were fully functional on Linux in 1999. IBM believed that JFS was a strong technology and could add value to the Linux operating system.

Contacts were made with the top Linux filesystem developers and the possibility of adding yet another journaling filesystem was explored. One of the basic underlying philosophies of Linux is that choice is good, so the idea of another journaling filesystem was accepted.

IBM started moving JFS to Linux in December 1999, and by February 2000, they had released the first source code. This initial release contained the reference source code, the mount/unmount function and support for the ls command on a JFS partition.

Installing JFS on a Separate Partition

JFS has been incorporated into the 2.5.6 Linux kernel and also is included in Alan Cox's 2.4.X-ac kernels, beginning with the February 2002 release of 2.4.18-pre9-ac4. Alan's patches for the 2.4.x series are available from kernel.org. You also can download a 2.4 kernel source tree and add the JFS patches to this tree. JFS comes as a patch for several of the 2.4.x kernels, so get the latest kernel from kernel.org.

At the time of this writing, the latest kernel is 2.4.18 and the latest release of JFS is 1.0.20. We use them in the following section. The JFS patch is available from the JFS web site. You also need both the utilities (jfsutils-1.0.20.tar.gz) and the filesystem (jfs-2.4.18-patch and jfs-2.4-1.0.20.tar.gz) patches. Several Linux distributions are already shipping JFS: Turbolinux, Mandrake, SuSE, Red Hat and Slackware all ship JFS in their latest releases.

Patching the Kernel to Support JFS

If you use any of the previously named distributions, you do not need to patch the kernel for the JFS code. You need only to compile the kernel to support JFS (either as built-in or as a module).

First, download the standard Linux kernel. If you have a /usr/src/linux directory, move it, so it won't replaced by the linux-2.4.18 source tree. After you download the kernel, named linux-2.4.18.tar.gz, save it under /usr/src and untar it. This operation will create a new /usr/src/linux directory.

The next step is to get the JFS utilities and the appropriate patch for kernel 2.4.18. Create a directory for JFS source, /usr/src/jfs1020, and download to that directory the JFS kernel patch, the jfs-2.4-18-patch and the JFS patches, jfs-2.4-1.0.20.tar.gz. At this point, you have all the files needed to patch the kernel.

Next, change to the directory of the kernel 2.4.18 source tree to apply the JFS kernel patch:

```
% cd /usr/src/linux
% patch -p1 < /usr/src/jfs1020/jfs-2.4-18-patch
% cp /usr/src/jfs1020/jfs-2.4-1.0.20.tar.gz .
% tar zxvf jfs-2.4-1.0.20.tar.gz
```

Configure the kernel and enable JFS by going to the Filesystems section of the configuration menu and enabling JFS support, **CONFIG_JFS_FS=y**. You also have the option to configure JFS as a module. In this case you need only to recompile and re-install kernel modules:

```
% make modules && make install_modules
```

Otherwise, if you configured the JFS option as kernel built-in, you need to recompile the kernel (in /usr/src/linux):

```
   % make dep && make clean && make bzImage
```

Then, recompile and install modules (only if you added other options as modules):

```
   % make modules && make modules_install
```

Finally, install the new kernel:

```
   % cp arch/i386/boot/bzImage /boot/bzImage-jfs
   % cp System.map /boot/System.map-jfs
   % ln -s /boot/System.map-jfs /boot/System.map
```

Don't forget to run lilo. (If you have never recompiled the kernel, read the Kernel-HOWTO to learn how.)

After you compile and install the kernel, you should compile and install the JFS utilities. Save the jfsutils-1.0.20.tar.gz file into /usr/src/jfs1020 and then:

```
   % tar zxvf jfsutils-1.0.20.tar.gz
   % cd jfsutils-1.0.20
   % ./configure
   % make && make install
```

Having built and installed the JFS utilities, the next step is to create a JFS partition. In the following example, we use a spare partition; the next section will demonstrate how to migrate an existing partition into JFS.

If there is unpartitioned space on the disk, you can create a partition using fdisk. In our test system, we had /dev/hdb3 as a spare partition, so we formatted it as a JFS partition. After the partition is created, reboot the system to make sure the new partition is able to create the JFS.

To create the JFS, apply the following command:

```
   % mkfs.jfs /dev/hdb3
```

After the filesystem has been created, you need to mount it. To get a mount point, create a new empty directory, such as /mnt/jfs, and use the following mount command:

```
   % mount -t jfs /dev/hdb3 /mnt/jfs
```

When the filesystem is mounted, you are ready to try out JFS.

To unmount JFS, simply use the umount command with the same mount point as before:

```
   % umount /mnt/jfs
```

Migrating Your Partition from ext2 to JFS

In the previous section, we explained how to create a JFS filesystem using an existing spare partition. Now, we demonstrate how to migrate your current system from another filesystem, such as ext2, to JFS. We look at how to introduce a JFS partition to your Linux configuration. In a second step, we make that partition the root filesystem.

What partition scheme do you need to create a JFS root partition? The migration process requires an empty partition. Let's assume that /dev/hda5 is the current root partition and that it uses ext2. We use /dev/hda6, which is our empty partition, as our JFS root partition. This partition needs to be of equal or larger size than the current root partition. The ext2 partition will be duplicated on the JFS partition. Afterward, if you do not wish to keep the ext2 partition, you can reformat it without losing your Linux system.

In order to create a root JFS partition on /dev/hda6, follow the instructions mentioned earlier to get support for JFS in the kernel. To make this partition a bootable partition for Linux, you need to reproduce a complete Linux installation. A simple way of doing so is to copy all the files to the JFS partition. First, mount the filesystem:

```
% mount -t jfs /dev/hda6 /jfs
```

Then, copy all files from ext2 filesystem to the JFS partition:

```
% cd /
% cp -a bin etc lib boot dev home usr var [...] /jfs
  You need special handling for /proc and /tmp:
        % mkdir /jfs/proc
        % chmod 555 /jfs/proc
        % mkdir /jfs/tmp
        % chmod 1777 /jfs/tmp
```

It is important to create /proc and /tmp with the correct permissions. Permission 1777 means the only people who can rename or remove any file in that directory are the file's owner, the directory's owner and the superuser. The last steps involve changing /etc/lilo.conf and /etc/fstab. First, we change lilo.conf to boot using the kernel on our JFS partition. Notice that root is different from the first entry we made as well as from the label. Thus, the image to be booted will not be found in /dev/hda5/boot, but in /dev/hda6/boot:

```
image=/boot/vmlinuz-jfs
        label=jfs-kernel
        read-only
        root=/dev/hda6
```

Finally, we need to change /jfs/etc/fstab to tell the Linux system what filesystem it is using. Change the following line:

```
LABEL=/         /       ext2    defaults        1 1
```

so that it says:

```
   /dev/hda6        /       jfs     defaults        1 1
```

Now, you can reboot and choose jfs-kernel. This will start Linux with the JFS root filesystem.

After a crash, the log replay occurs automatically. Instead of the usual fsck messages, you should see JFS journaling messages. Replaying the log is necessary when a filesystem becomes unstable.

The JFS Experience at the Ericsson Research Lab in Montréal

One of the responsibilities of the Open Systems Lab at Ericsson Research is to design, implement and benchmark carrier-class platforms that run telecom applications. These carrier-class platforms have strict requirements regarding scalability, reliability and high availability. They must operate nonstop, regardless of hardware or software errors, and they must allow operators to upgrade hardware and software during operation without disturbing the applications that run on them. As a result, they must offer extreme reliability and high availability, often referred to as a five-nines availability (99.999% uptime).

To maintain such availability, these carrier-class platforms were designed with many features that allow software to be upgraded while the system is running and providing service. These features include fault tolerance implemented in the software, network redundancy to handle catastrophic situations such as earthquakes and in-service upgrade features.

Although many precautions have been taken to protect the system, there is always a remote chance that the processor (or server node) will fail. Thus, as a last resort, we need to reboot the processor. In this extreme case, we need to be able to reboot the processor and bring it to normal status, serving requests as soon as possible, with a minimal downtime.

Our interest in journaling filesystems for the carrier-class Linux platform came from the fact that these filesystems provide a fast filesystem restart. In the case of a system crash, a journaling filesystem can restore a filesystem to a consistent state more quickly and more reliably than other filesystems.

Initially, we started to experiment with the IBM JFS in early 2000. The JFS team was helpful and supportive, and their representative, Steve Best, visited our lab in January 2001. Since then, we have been following JFS development closely and upgrading our servers to support the latest versions.

Figure 1. 1U Rackmount Units Used to Test JFS

The first installations of JFS were done on 1U rackmount units with Celeron 500MHz processors, 256MB of RAM and 20GB IDE disks. These units provided us with a working environment to test JFS and to experiment with its features using some of our applications. Since JFS version 1.0.0 was released in June 2001, we decided to install JFS on our test Linux platform, shown in Figure 2.

Figure 2. Telecom-Grade HW Used to Test JFS on Linux

Our Linux systems are designed to serve short transaction-based requests. JFS provides a log-based, byte-level filesystem targeted for transaction-oriented systems, which makes it quite suitable for our type of systems.

The advantages of JFS from a telecom point of view are that it provides improved structural consistency, recoverability and much faster restart times than non-journaling filesystems, such as traditional UNIX filesystems. In most cases, the other filesystems are subject to corruption in the event of a system crash. They rely on restart-time utilities like fsck, which examines all of the filesystem's metadata to detect and repair structural integrity problems. This is a time-consuming and error-prone process; in a worst-case scenario, it can lose or misplace data. Telecom platforms cannot afford a process that prolongs a system's downtime.

With JFS, in case of a system failure, a filesystem is restored to a consistent state by replaying the log and applying log records for the appropriate transactions. The recovery time associated with this log-based approach is much faster, because the replay utility examines only the log records produced by recent filesystem activity, rather than examining all filesystem metadata.

Requirements for a Journaling Filesystem

Conclusion

JFS is a key technology for servers because it provides fast filesystem restart times in the event of a system crash. The JFS team's most important goal is to create a reliable, high-performance filesystem. The JFS team is making great progress in porting JFS to Linux. From a performance point of view and based on the various published benchmarks, JFS comes out as a winner. To get involved, visit the JFS Project page on developerWorks.

Acknowledgements

The Open Systems Lab at Ericsson Research for supporting our work with Linux and open-source software.

Resources

Steve Best (sbest@us.ibm.com) works in the Linux Technology Center of IBM in Austin, Texas. He is currently working on the Journaled Filesystem for Linux Project. Steve has done extensive work in operating system development with a focus in the areas of filesystems, internationalization and security.

David Gordon (gordd00@dmi.usherb.ca) is finishing his Bachelor's degree in Computer Science at Sherbrooke University in Québec, Canada. He is a co-op student with the Ericsson Research Lab in Montréal.

Ibrahim Haddad (Ibrahim.Haddad@Ericsson.com) is a researcher at the Ericsson Corporate Unit of Research in Montréal, Canada. He is involved with the system architecture of third-generation wireless IP networks. Ibrahim represents Ericsson on the Technical Sub-Groups of the Open Source Development Lab (OSDL). He is currently a DrSc Candidate at Concordia University.

OpenACS Templates

Reuven M. Lerner

Issue #105, January 2003

Reuven explains how OpenACS templates collect and return data and perform automatic error checking.

Over the last few months, we've looked at the Open Architecture Community System, an open-source toolkit for creating community web sites. Last month, we even looked at how we can create a simple application package using the ArsDigita Package Manager (APM).

But at its heart, web development is all about receiving inputs in HTTP GET and POST requests and about producing pages of HTML in response to those requests. Each web development toolkit has its own set of templates, and each type of template has its own personality and quirks.

This month, we take a closer look at the OpenACS templating system, which is similar in some ways to Zope Page Templates (ZPT). The OpenACS templates are rather sophisticated in the way they collect and return data and make it possible to perform many types of automatic error checking that would otherwise be tedious or simply ignored.

OpenACS Templates

Older versions of OpenACS used a templating system known as AOLserver Dynamic Pages (ADPs), similar to JSP, ASP or PHP. ADPs could contain Tcl code, thanks to the multithreaded Tcl interpreter running inside of AOLserver, the HTTP server that powers OpenACS. However, ADP had a number of problems. Each Tcl block was evaluated independently, making it difficult to have conditional code inside of a template. There was no standard way to ensure that ADPs were passed required parameters to give parameters optional values. And of course, the troubles really began when developers and designers needed to work on the same file. Moreover, much of OpenACS was written in simple .tcl files, which can be quite daunting for a nonprogrammer.

The programmers at ArsDigita, whose code lives on in the OpenACS Project, decided that a paradigm shift was in order. No longer would pages be invoked with their familiar .html and .adp suffixes; instead, they would be called without a suffix at all.

This is possible because of AOLserver's willingness to search, in order, for an appropriate page. Given the URL /foo, AOLserver will first look for /foo.tcl, then for /foo.adp and finally for /foo.html. (This configuration setting can be changed in the nsd.tcl configuration file that typically is located in /usr/local/aolserver.)

The OpenACS templating system relies on this fact to split the work between two different files. In general, the output generated by an HTTP request has to go through two different files. The .tcl file executes first, performing database queries and setting variables. Its final line is typically going to be ad_return_template, a Tcl procedure that then invokes the companion .adp page. The ADP can retrieve the variables as data sources.

Because the .tcl and .adp files are supposed to be developed by separate people, it's natural to expect them to drift apart or have compatibility issues. The ArsDigita engineers avoided this problem by setting up a “page contract”, meaning a list of HTTP parameters that the .tcl file expects to receive, and then a list of Tcl variables (known as data sources) that will be available to the .adp page in its display.

Data Sources

Data sources are simply Tcl variables by another name. Inside of an .adp page, you specify a data source like @this@, with @ signs around the name. If the data source has not been defined, the template will exit with a Tcl stack trace, complaining of an unknown variable.

Data sources can be placed anywhere in a file. Their values are substituted in place of those on the HTML page before the HTML is sent to the user's browser, which means that data sources can define not only text, but also image names and stylesheet attributes.

For example, here is a simple OpenACS template that displays the user's first name in a first-level headline:

```
<master>
    <h2>@first_name@</h2>
    <p>That's you, isn't it?</p>
</master>
```

The master tags indicate that the page in question is not complete HTML output, but rather is meant to be inserted inside of the local master (i.e., a template pair named master.tcl/master.adp). The local master, in turn, is wrapped inside of the site's default master (normally in a template pair named default-master.tcl and default-master.adp, but this is configurable using parameters). Thus, the resulting page consists of:

```
default master top
    local master top
        our page
    local master bottom
default master bottom
```

In this way, you can create sites that have a unified look and feel, with common headers and footers, such as menubars and contact information. The notion of masters and default masters is similar in many ways to autohandlers in the Perl-based HTML::Mason templating system.

This is all well and good, but where is @first_name@ defined? It is defined in the companion .tcl file. But it's not enough for the .tcl file to set the first_name variable. It also must mark the variable for export as a data source by naming it explicitly.

.tcl pages name their inputs and outputs in arguments to the ad_page_contract procedure, which normally executes at the top of a page. ad_page_contract is a powerful mechanism that lets us create pages that expect to receive certain inputs, which in turn promise to produce certain outputs. A call to ad_page_contract can range from extremely simple to extremely complex, depending on the needs of the .adp template page. For example, a .tcl page that sets the user's first name to a dummy value and then exports it to the .adp page could look like:

```
ad_page_contract {
    Comments and CVS information go here.
} {
} -properties {
    first_name:onevalue
}
set first_name "Dummy first name"
ad_return_template
```

The call to ad_page_contract tells the templating system that the variable first_name will be exported. We then set the variable's value and finally pass those values to the template with ad_return_template. (For unfortunate historical reasons, data sources are passed in the -properties parameter, rather than something with a more informative name.)

We can pass multiple variables to the template by naming additional variables in ad_page_contract:

```
ad_page_contract {
    Comments and CVS information go here.
} {
} -properties {
    first_name:onevalue
    last_name:onevalue
}
set first_name "FirstName"
set last_name "LastName"
ad_return_template
```

Lists and Multirows

As you can see, ad_page_contract makes it easy to pass individual text strings to the template. But in many cases, particularly when retrieving results from a database, we want to pass lists of values. OpenACS templates take care of this without any problem. For example, the following .tcl page retrieves the list of users on the system and places that in a “list” data source:

```
ad_page_contract {
    Comments and CVS information go here.
} {
    users:onelist
}

set users [db_list get_all_users {
    SELECT PE.first_names || ' ' ||
           PE.last_name as users
      FROM Parties PA, Persons PE
     WHERE PA.party_id = PE.person_id
  ORDER BY PE.last_name, PE.first_Names }]
 ad_return_template
```

As you can see, our SQL query returns a single column, named users. The OpenACS database API turns this into the Tcl list users, which we then export as the users' data source. But because we export it with the :onelist descriptor, an .adp page can iterate over each individual element:

```
<master>
     <list name="users">
         <li> @users:item@
     </list>
</master>
```

Our iterator is <list>, the contents of which execute once for each element in the list. The current element is available as @users:item@; the number of the current iteration is available as @users:rownum@, and the current iteration is available as @users.

If you want to iterate over multiple database rows, your .tcl page can export a multirow. A multirow contains all of the rows that were returned, with column names. Here's the Tcl side of things:

```
ad_page_contract {
} {
} -properties {
    users:multirow
}
db_multirow users get_info "
    SELECT PE.first_names || ' ' ||
           PE.last_name as name,
           PA.email FROM Parties PA, Persons PE
     WHERE PA.party_id = PE.person_id
```

```
  ORDER BY PE.last_name, PE.first_names"
ad_return_template
```

The db_multirow procedure takes three arguments: the name of an array that will be populated (and exported as a data source), the name of the query (which is used in conjunction with database-independent .xql files) and the fallback query that is used if no .xql file is found. In the .adp template, we can then do the following:

```
<master>
  <ul>
  <multiple name="users">
    <li>
    <a href="mailto:@users.email@"
    @users.name@</a>
   </multiple>
   </ul>
</master>
```

There are two tricky things to note here. First, the iterating tag is multiple, even though the data source is exported as a multirow. Using the wrong name in the wrong place can create hard-to-understand bugs. More subtly, the element selector within a <multiple> tag is a period (@users.email@), while it's a colon (@users:item@) in a <list> tag. I find myself constantly making mistakes on this and checking previously working pages of code to ensure that I use the right syntax with the right page.

Inputs

So far, we have only looked at ways in which ad_page_contract allows us to export data from the .tcl page to the .adp page. But .tcl pages can accept inputs as well, via either GET or POST requests. ad_page_contract allows us to specify which inputs we expect to receive, to assign default values as necessary and to check that the inputs are in a particular format:

```
ad_page_contract {
} {
    foo
} -properties {
    foo2:onevalue
}
set foo2 "$foo$foo"
ad_return_template
```

In the above example, the .tcl page expects to receive a parameter (via either GET or POST) named foo. The parameter's value is then used in the creation of a new data source, foo2, which contains a doubled version of foo.

If someone invokes the above page without passing a foo parameter, OpenACS automatically produces an error message that looks like the following:

```
We had a problem processing your entry:
* You must supply a value for foo
Please back up using your browser, correct it,
and resubmit your entry.
Thank you.
```

We can give foo a default value if it is not passed, by making it the first element of a Tcl list and giving a default value as the second element:

```
ad_page_contract {
} {
    {foo "blah"}
} -properties {
    foo2:onevalue
}
set foo2 "$foo$foo"
ad_return_template
```

So invoking this page with a parameter of foo=abc will produce output of “abcabc”, and invoking it without any parameter will produce output of “blahblah”.

We can add one or more options to each parameter to limit the types of information that we receive. For example, we can trim any leading or trailing whitespace from an input parameter or ensure that we only receive an integer greater than zero:

```
ad_page_contract {
} {
    sometext:trim
    anumber:naturalnum
}
```

You can use multiple options on a parameter by separating them with commas:

```
ad_page_contract {
} {
    sometext:trim,nohtml
    {anumber:naturalnum 50}
}
```

The above page contract says that anumber must be a natural number, with a default value of 50 if nothing is specified. sometext will be trimmed for leading and trailing whitespace but may not contain any HTML tags. There are related html and allhtml options that allow safe HTML tags and any HTML tags, respectively.

Page contracts can get even fancier. For example, you can create a date selection widget with the ad_dateentrywidget function. So you can imagine a .tcl page that looks like:

```
ad_page_contract {
} {
} -properties {
    datewidget:onevalue
}
set datewidget [ad_dateentrywidget datewidget]
ad_return_template
```

The accompanying .adp page, which will display this date widget, then looks like:

```
<master>
    <form method="POST" action="date-2">
    @datewidget@
    <input type="submit" value="Send the
date">
    </form>
</master>
```

In other words, our HTML form will send the contents of the date widget to date-2, a .tcl page that will display its results in an .adp page. date-2.tcl could tell ad_page_contract that the incoming datewidget parameter will contain a simple array. But, we additionally can declare datewidget to be a parameter of type **date**, which automatically gives us four array elements:

```
ad_page_contract {
} {
  datewidget:array,date
} -properties {
  date_month:onevalue
  date_day:onevalue
  date_year:onevalue
  date_full:onevalue
}
  set date_month $datewidget(month)
  set date_day $datewidget(day)
  set date_year $datewidget(year)
  set date_full $datewidget(date)
  ad_return_template
```

Our .adp page, date-2, can now display the date information in a variety of formats:

```
<master>
    <p>Month: @date_month@</p>
    <p>Day: @date_day@</p>
    <p>Year: @date_year@</p>
    <p>Full text: @date_full@</p>
</master>
```

Note that the full version of the date widget is perfectly acceptable for SQL queries. This comes in handy when entering dates or when using them in comparison queries.

Other Functions

We have only scratched the surface of the OpenACS templating system. ad_page_contract additionally supports verification routines that allow you to check multiple parameters or to signal errors based on information found in the database. You actually can define your own custom error messages that may appear in the case of trouble. Even outside of ad_page_contract, a .tcl page also can call the universal ad_return_complaint function, which produces error messages in nicely formatted HTML pages.

In addition, the OpenACS templating system has a complete set of form-building routines that allow a programmer to specify an HTML form using Tcl procedures. The contents of the form then can be exported to the .adp page using data sources. Not only does the form builder cut down on the amount of HTML you have to write, but it makes it easy to create a two-stage form submission process, in which users get a chance to preview their work before sending it in.

Finally, OpenACS templates include a number of additional tags, such as <if>, that allow you to include text and images conditionally, depending on the values of other data sources.

Conclusion

While OpenACS is often touted as a remarkable system because of its elaborate data model and advanced applications, I have found the templates to be one of the more compelling parts of OpenACS. The graphic designers with whom I work enjoy the separation between .tcl and .adp pages, and I like that I can check for errors and pass multiple values without having to remember that there is no obvious connection between them at the HTTP level.

While the learning curve for OpenACS can be quite steep, learning how the templates work is both a gratifying and interesting way to start with this system. Given the many application packages that come with OpenACS, there also are numerous examples of the templates right in the code after you download the system.

Resources

email: reuven@lerner.co.il

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

When I'm Calling You…on Video

Marcel Gagné

Issue #105, January 2003

Have we finally reached one of the promises of the future? GnomeMeeting presents a start.

Do you remember *2001: A Space Odyssey*, François? As we welcome 2003, I can't help but think about all the promises that vision of the future held, and it makes me a little sad, *mon ami*. Ah, *merci*. The 1998 Rhone Hermitage is exactly what I need. I realize that our guests will be here shortly, François, but consider this. We have a space station, but it's not quite the majestic wheel in space we see in the movie. There's certainly no lunar base nor orbital hotel, but at least we have this.

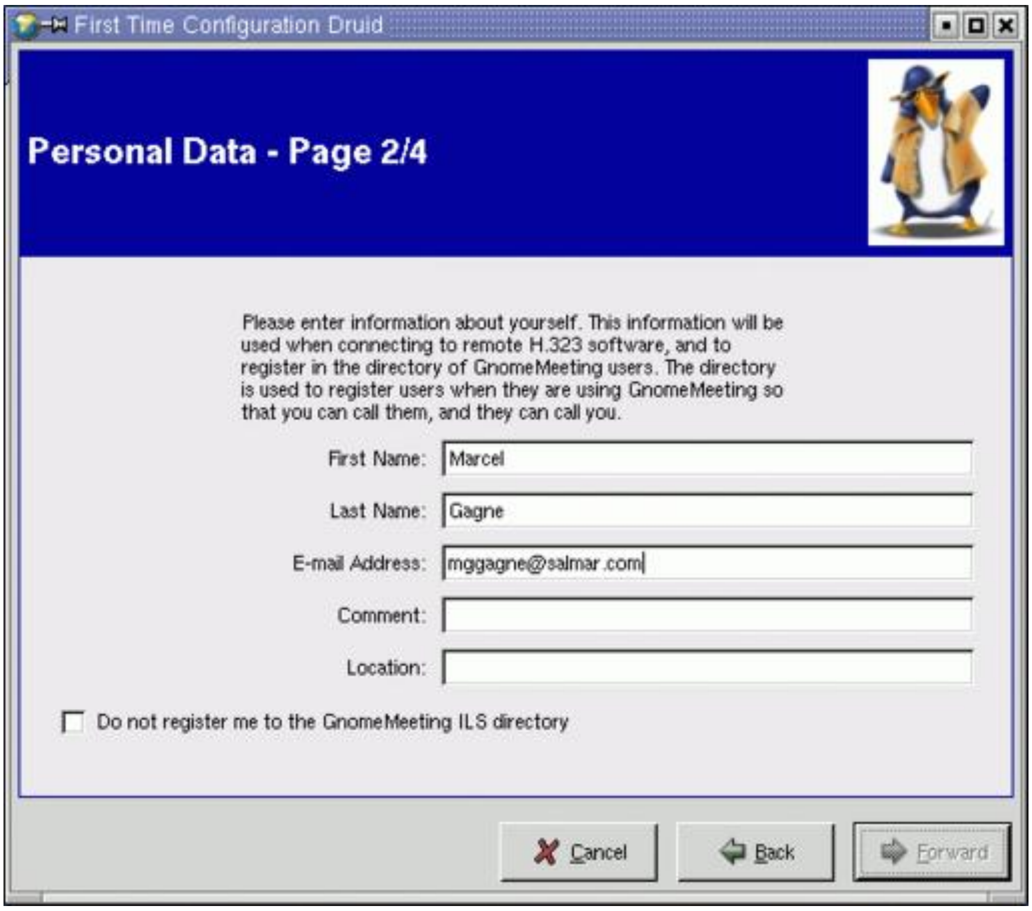
Quoi? Why it's a video phone, François, and today, I will demonstrate it to our guests. What did you say? Ah, but they are already here! Welcome once again, *mes amis*, to *Chez Marcel*, where fine Linux cooking meets fine wine, *non*? François has your tables ready and has already opened the wine.

From time to time, in this restaurant, we have offered up recipes that make use of your webcam, from capture utilities to home security. Today, we visit the webcam once again to bring you something that was promised so many years ago. When I was but a small child, it seemed that every television science-fiction program was promising a video phone. Years passed, and while I have seen such things demonstrated on television, my video phone remained as distant as the faraway studios themselves.

Since opening this restaurant, I have had the pleasure of discovering that many of my childhood dreams have become possible when cooking with Linux. So it is for the video phone, or at least, a passable incarnation of it: Damien Sandras' GnomeMeeting. GnomeMeeting even will work with Microsoft Netmeeting so you can talk to your friends running that company's OS. Finally, don't let the name fool you. GnomeMeeting works extremely well with KDE and docks nicely into the panel.

A number of distributions come with GnomeMeeting and the required libraries (pwlib and openh323) on the CD, though not necessarily installed. The www.gnomemeeting.org site does have packages for the major distributions (RPMs and DEBs) as well as source. You should certainly look there first. OpenH323 depends on PwLib, so you will need to install that. On both my Mandrake and Red Hat systems, I installed it using the RPMs. I should point out that while you can build GnomeMeeting from source, the OpenH323 libraries in particular can be difficult (not to mention lengthy). If you can use the binaries for this one, I would recommend it.

When starting GnomeMeeting for the first time, you'll be presented with the First Time Configuration Druid. Part of this process involves registering with the directory of GnomeMeeting users—think of it as a large on-line telephone directory. You can opt not to be listed by clicking the check box at the bottom of the Druid's registration box. When you are happy with the information you are presenting, click the Forward button to continue, and you will be asked to specify the connection type you are using.



Please enter information about yourself. This information will be used when connecting to remote H.323 software, and to register in the directory of GnomeMeeting users. The directory is used to register users when they are using GnomeMeeting so that you can call them, and they can call you.

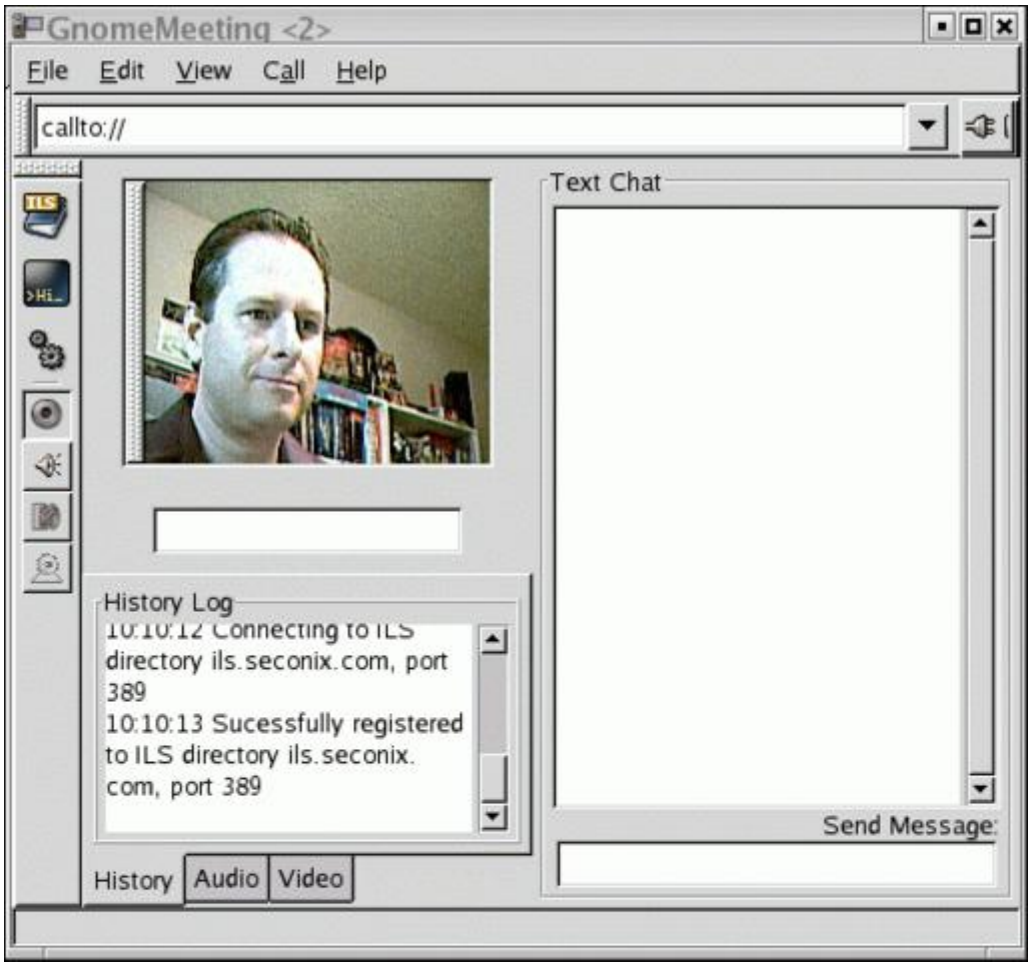
First Name:	Marcel
Last Name:	Gagne
E-mail Address:	mggagne@salmar.com
Comment:	
Location:	

☐ Do not register me to the GnomeMeeting ILS directory

Figure 1. Setting up GnomeMeeting is a breeze.

Click Apply on the final screen and GnomeMeeting fires up. You can configure several options with the package by clicking Edit on the menubar and selecting Preferences. You also can rerun the Druid at any time. The first preference I set was under the Video Devices section. When GnomeMeeting started up, I wanted the video preview to be on and the video size set to large.

To place a call to another PC on my local LAN, I type **callto://192.168.22.2** on the GnomeMeeting location field (right below the menubar). A little pop-up window appears on the second PC warning the user of an incoming connection. If you accept the connection, the two clients will be able to communicate.

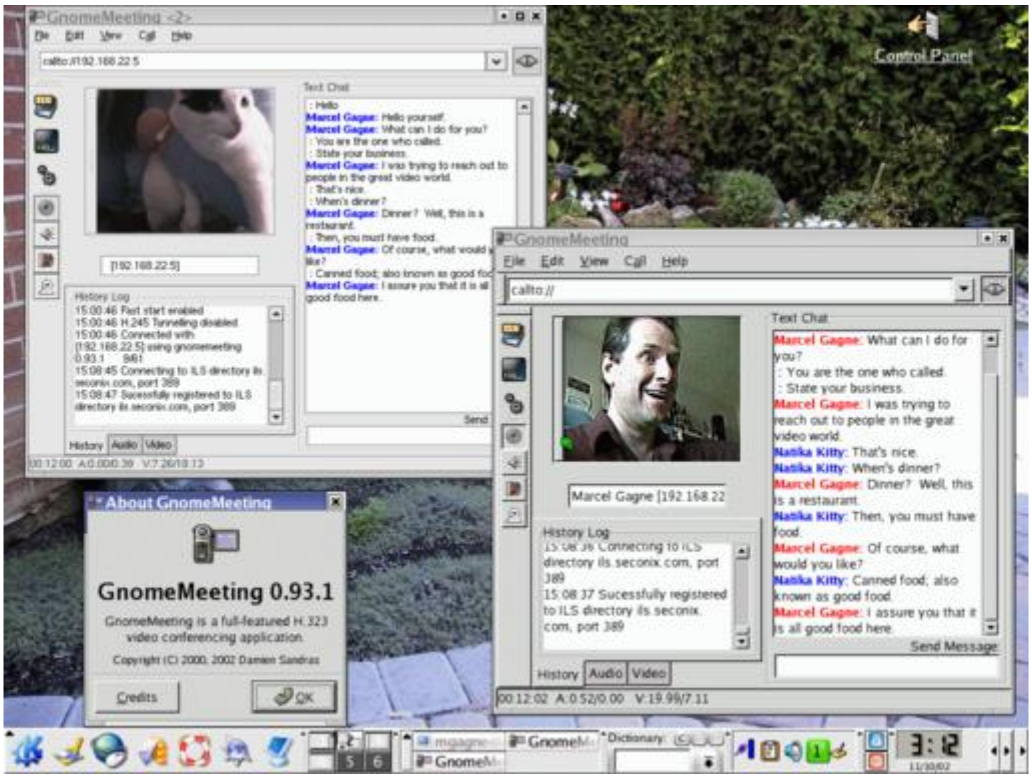


```
10:10:12 Connecting to ILS
directory ils.seconix.com, port
389
10:10:13 Sucessfully registered
to ILS directory ils.seconix.
com, port 389
```

Figure 2. Ready to Call the World with GnomeMeeting

Notice the button bar to the left in Figure 2. You can turn your video or audio on and off, and you can bring up a chat window for text message exchange. If you have a microphone, GnomeMeeting will make use of that as well. In fact, GnomeMeeting will work under a number of different configurations. You can run video only, audio only, text only or any combination of the three modes. Of course, it can be a little disconcerting to know that somebody out there can see you, but you can't see them. *Mon Dieu*! Now I definitely need a refill. François, bring the wine. *Vite*!

When you run GnomeMeeting, make sure you turn on the control panel. It opens up to a tabbed window in the application providing support for audio and video controls as well as a history window. This shows the status of calls, your registration with on-line directories and other information and can be turned on or off at any time without affecting the transmission. Figure 3 handily displays a desktop with two sessions running (it would appear that I managed to reach a *cat*).



: Hello
Marcel Gagne: Hello yourself.
Marcel Gagne: What can I do for you?
: You are the one who called.
: State your business.
Marcel Gagne: I was trying to reach out to people in the great video world.
: That's nice.
: When's dinner?
Marcel Gagne: Dinner? Well, this is a restaurant.
: Then, you must have food.
Marcel Gagne: Of course, what would you like?
: Canned food; also known as good food.
Marcel Gagne: I assure you that it is all good food here.

15:00:46 Fast start enabled
15:00:46 H.245 Tunneling disabled
15:00:46 Connected with [192.168.22.5] using gnomemeeting 0.93.1 9/61
15:08:45 Connecting to ILS directory ils.seconix.com, port 389
15:08:47 Sucessfully registered to ILS directory ils.seconix.com, port 389

Marcel Gagne: What can I do for you?
: You are the one who called.
: State your business.
Marcel Gagne: I was trying to reach out to people in the great video world.
Natika Kitty: That's nice.
Natika Kitty: When's dinner?
Marcel Gagne: Dinner? Well, this is a restaurant.
Natika Kitty: Then, you must have food.
Marcel Gagne: Of course, what would you like?
Natika Kitty: Canned food; also known as good food.
Marcel Gagne: I assure you that it is all good food here.

15:08:36 Connecting to ILS directory ils.seconix.com, port 389
15:08:37 Sucessfully registered to ILS directory ils.seconix.com, port 389

Figure 3. Somebody Is Calling for Dinner

On the space station, the call was a long-distance one: from space to a little girl on Earth. While chatting inside our offices is fine, what about the outside world? Who are you going to call? The official telephone directory for GnomeMeeting is available at ils.seconix.com. In order to browse the directory and find other users, you also must be registered. Start GnomeMeeting, click on the directory icon to the left, and search for another party—simple...almost.

You see, if you are running GnomeMeeting on your corporate or home LAN, you should have no problems. The same holds true if you are running it from a single machine connected to the Internet—odds are this will work without a hitch. The catch, *mes amis*, comes when you try to work from behind a masqueraded (or NATed) firewall. The ports you need to allow through are as follows:

```
TCP ports 1720 and 30000 thru 30010
UDP ports 5000:5003
```

At this stage of the game, you can get into some reasonably complex firewall issues. I'll get to that shortly, but there is a simple approach called RSIP for Linux, a simple alternative to NAT. RSIP is a relatively new protocol that, like

NAT, allows you to share a single connection between multiple clients. You can surf, send e-mail and whatever else happens to fall into the range of client services. You can also use it to redirect ports easily on a single server behind the firewall without modifying the packets in any way. The only way you can do that with NAT or masquerading is with the use of loadable modules. These modules are, at best, experimental when it comes to H323 and our old friend GnomeMeeting. RSIP solves that problem, which doesn't mean you can throw away your iptables—RSIP still makes use of them.

Start by heading over to openresources.info.ucl.ac.be/rsip/index.php and picking up the latest source. Prebuilt binaries are available for some releases—if yours isn't there, never fear; this is an easy build:

```
tar -xzvf rsipd-0.9.3.src.tar.gz
cd rsipd-0.9.3
make
su -c "make install"
```

That's the server side. You need a client as well, and at this time, there are two versions: one in Python and a kernel module called krsip. At the time I prepared this menu, the module was considered safe for testing, and the Python client was considered stable. Building the module is simply a matter of extracting the source and doing a **make install**. The catch is that only kernels 2.4.18 and 2.4.19 are supported. As for the Python client, there's no building required.

To use RSIP for GnomeMeeting, start by editing the /etc/rsip/rsipd.conf file and modifying a few relevant parameters. These are the eternal and internal interfaces of your firewall, as well as a network pool to be served by the server. I've allocated the first ten addresses on my internal 192.168.22.0 network:

```
EXT_IF eth1
INT_IF eth0
POOL 192.168.22.1-10
```

Now, start the server using the script created by the install:

```
/etc/init.d/rsipd start
```

To use the Python client, execute the following (note that this is one long, unbroken line. The IP address specified by the -s parameter is that of the server):

```
python /path_to/rsipclient-0.19.py \
-d -v -s 192.168.22.10 \
-l 1720,5000,5001,5002,5003,30000,30001,30002,30003,
30004,30005,30006,30007,30008,30009,30010
```

I should probably note that the client runs on the *client* PC, *non*? You also must run it as root. The beauty is that you can continue to use your existing firewall (iptables) rules. Using the kernel module is even easier. It requires only that you load it while specifying the address of your server:

```
insmos krsip ip=192.168.22.10
```

As a test, I brought up the ILS search window by clicking on the icon to the left. This brings up the XDAP Server Browser where I clicked Refresh, chose someone more or less at random and double-clicked on the entry. Seconds later, I was chatting with someone in Paris, France from my home in Canada.

See, it is easy. You are now ready to share in the excitement of video phone communication, and it is only two years late.

Once again, *mes amis*, closing time approaches. At least this time, we will be able to chat via our futuristic video phones (even if 2001 was two years ago now). The future isn't what it used to be, wouldn't you agree? We'll talk about this again in 2010. Before I bid you all *Bonsoir*, François will refill your wineglasses a final time. While you sip that last glass, let's see who is awake in Finland, shall we? Until next month. *A votre santé*! *Bon appétit*!

Resources

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

An Introduction to FreeS/WAN, Part I

Mick Bauer

Issue #105, January 2003

VPN tunnels for secure wireless and WAN connections, Part I of II.

Over the past five years or so, IPSec has emerged as the leading standard for building encrypted virtual private network (VPN) connections. FreeS/WAN (www.freeswan.org), the free secure wide area network, is the most popular and one of the most mature free implementations of IPSec, and it runs exclusively on Linux systems. This month and next we're going to discuss why and how to use FreeS/WAN for secure network communications, starting with secure wireless networking.

VPN Basics

Until recently the two most common uses for VPNs were network-to-network (site-to-site) connections and remote-access solutions. In site-to-site connections (Figure 1), each network/site has a VPN gateway, a VPN server that communicates to other VPN gateways via IPSec (or other VPN protocol) tunnels. It also acts as a router for hosts on its local network that need to send packets to other connected VPN sites. In other words, in a site-to-site VPN, multiple users or hosts share a single tunnel to communicate to multiple hosts on a remote network.

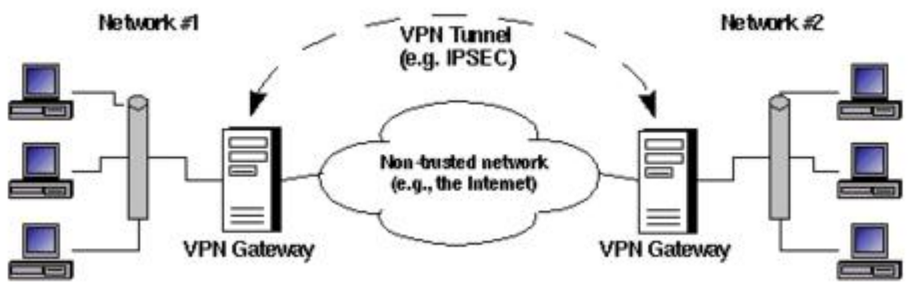


Figure 1. Site-to-Site VPN

Remote-access VPNs, including the kind used over wireless LANs, are slightly different. Rather than connecting an entire network to some other network, a remote-access VPN tunnel connects a single user or computer to a remote network (Figure 2). Typically, the user's local VPN gateway is simply a software application running on her local system (the remote VPN gateway is usually a firewall or dedicated VPN device on the home network).

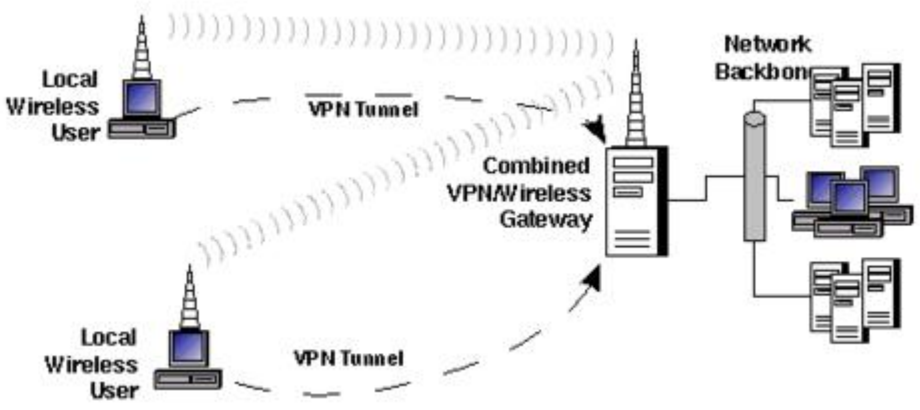


Figure 2. Wireless (Remote-Access) VPN

Wireless local area network (LAN) VPNs are an important subcategory of remote-access VPNs. Wireless networks are increasingly popular due to their convenience and low cost. However, by definition they broadcast all packets over radio waves, so it's easy to eavesdrop on them. Network vendors made a feeble attempt to provide wired equivalent privacy by creating the wireless encryption standard of the same name (WEP), but weaknesses in WEP's cryptographic implementations have rendered it prematurely obsolete. Therefore, many organizations that use wireless LANs leave WEP turned off. Instead they use VPN tunnels to encrypt wireless links.

Returning to Figure 2, note that a single system can serve as a combined VPN/ wireless gateway. Figure 3 shows an equally valid topology: the wireless and VPN gateways are separate devices.

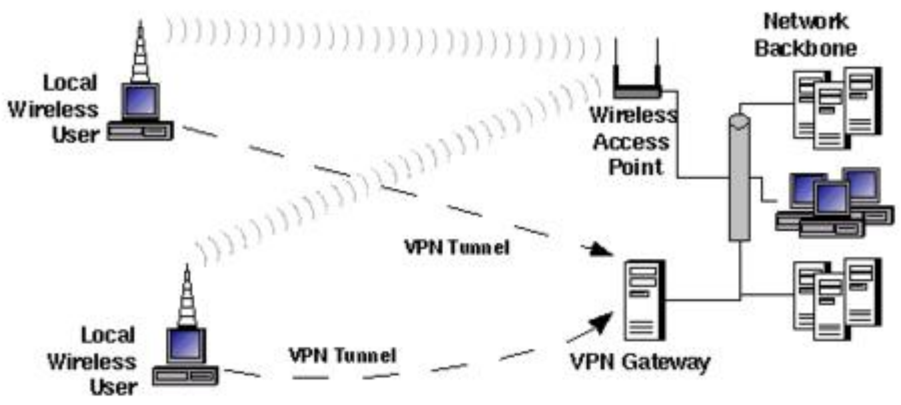


Figure 3. Another Wireless LAN VPN Topology

IPSec and FreeS/WAN

As I stated earlier, IPSec is the most popular VPN protocol. Because it's an extension of the IP protocol, it's the “official” VPN protocol of the Internet. For almost as long as IPSec has existed, John Gilmore and the FreeS/WAN Project team have been doing their best to promote IPSec's widespread adoption by developing and giving away the FreeS/WAN package for Linux. For definitive information about and the latest version of FreeS/WAN, see their home page at www.freeswan.org. Suffice it to say, FreeS/WAN is mature, well documented and well supported. If you run Linux, FreeS/WAN is *the* choice for your VPN needs.

Getting and Installing FreeS/WAN

Like Netfilter, FreeS/WAN consists of a kernel module that does the actual work and user interfaces that are used to configure it. Unlike Netfilter, FreeS/WAN is not included in standard Linux kernel sources and therefore is not part of the stock kernels in most Linux distributions. This is due to many countries' crypto export restrictions.

Retrofitting and even recompiling your kernel might sound like an unwieldy way to install FreeS/WAN. However, a number of Linux distributions, including SuSE, Debian and Mandrake, have FreeS/WAN packages that work with those distributions' stock kernels. For users of Red Hat 7.3, RPM packages of IPSec-enabled kernels (both binary and source) and the FreeS/WAN setup tools can be downloaded from Steamballoon at rpms.steamballoon.com/freeswan.

Because I personally run SuSE and Red Hat the most, I'll describe how to obtain and install FreeS/WAN for them. See the documentation at www.freeswan.org/doc.html if your needs are more complex. Depending on your kernel and distribution, you may have to compile FreeS/WAN from source, but this is well documented on the web site.

Installing FreeS/WAN on SuSE Systems

If you run SuSE with a stock kernel, simply install the package freeswan.rpm from the sec series. Make sure the kernel version of the ipsec.o module matches that of the SuSE kernel you run. The quick way to double-check your system's kernel version is with the command **uname -av**. To see the kernel version of your not-yet-installed freeswan.rpm package, use this command:

```
rpm -ql -p ./freeswan.rpm |grep ipsec.o
```

The kernel version will be indicated by the pathname to this file, e.g., /usr/lib/modules/2.2.18/ipv4/ipsec.o.

If the kernel versions match up, install the package with rpm, like this:

```
rpm -Uvh ./freeswan.rpm
```

Next, enable IPSec by opening /etc/rc.config and setting the variable START_IPSEC to “yes”.

Now it's time to replace the sample host key (RSA signing key pair) that was probably installed on your system by the FreeS/WAN RPMs. (If the creation date of /etc/ipsec.secrets is earlier than today's date, you need new keys.) The commands to do this for FreeS/WAN versions 1.92 and higher are:

```
mv /etc/ipsec.secrets /etc/ipsec.secrets.test
ipsec newhostkey --hostname
--output /etc/ipsec.secrets --bits 2192
```

You will, of course, replace *my.host.FQDN* with your host's fully qualified domain name, e.g., george.wiremonkeys.org.

For earlier versions of FreeS/WAN, use:

```
ipsec rsasigkey --hostname my.host.FQDN 2192 \
> /etc/ipsec.newkey
```

If you use the **ipsec rsasigkey** command, you'll also have to open /etc/ipsec.secrets with a text editor and replace everything between the curly brackets ({}) with the contents of ipsec.newkey (or whatever file to which you cat'd the new key).

Even though you haven't configured it yet, you may now test FreeS/WAN by starting and querying it:

```
/etc/init.d/ipsec start
ipsec whack --status
```

If the second command (**ipsec whack --status**) returns **000**, your FreeS/WAN installation is working properly.

Installing FreeS/WAN on Red Hat Systems

If you run Red Hat 7.3 with its stock kernel (version 2.4.18 as of this writing), download the appropriate IPSec-enabled kernel package from rpms.steamballoon.com/freeswan. You should grab the binary or source package for freeswan. Install the kernel package first.

If you've already got a stock kernel package installed (and it's almost certain that you do) you'll need to use the --force option, because Steamballoon's FreeS/WAN kernel package's base name (simply, kernel) is the same as that of the official Red Hat stock kernel. On my Celeron-based Red Hat 7.3 system, I installed the Steamballoon kernel package like this (all version numbers in examples are probably obsolete already):

```
rpm --force -i ./kernel-2.4.18-3ipsec.i686.rpm
```

Don't worry about forcing the install; the names of the new kernel's image file and module directory are unique and will *not* overwrite your old kernel. For example, on my Red Hat 7.3 system, the new image file and module directory are named /boot/vmlinuz-2.4.18-3ipsec and /lib/modules/2.4.18-3ipsec, respectively. Steamballon's RPM post-installation script overwrote the boot menu entry in /boot/grub/grub.conf for my old kernel with one for the new IPSec kernel (i.e., it removed the old kernel from the boot menu). But after I re-added a menu entry for the old kernel, I could choose either entry at boot time with no problem.

After you've installed the kernel package, install the user-space tools. On my system the command was:

```
rpm -i freeswan-1.97-0.i386.rpm
```

This RPM installs a startup script, /etc/init.d/ipsec, but does not enable it. You can do so like this:

```
chkconfig --add ipsec
```

Next, generate a new RSA signing key pair as described in the previous section. You can then start and test your FreeS/WAN installation, also described in the previous section.

Configuring a Wireless LAN VPN

This month, I have enough space to cover only one common FreeS/WAN scenario: wireless LAN tunneling, as illustrated in Figure 2. Figure 4 shows part of the network from Figure 2, this time with IP addresses. In this example, the wireless client system is dynamically IP addressed. As we'll see shortly, you don't need to know the IP addresses of all potential clients.

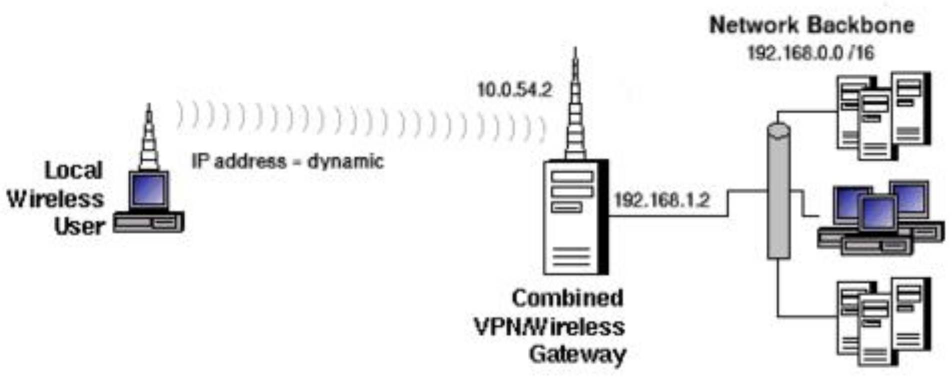


Figure 4. Wireless VPN with IP Addresses

The rest of this article depends on several important assumptions: 1) basic network connectivity is already in place, so wireless clients can connect to the Linux server; 2) basic packet forwarding works—the wireless clients can reach hosts on the other side of the wireless gateway; and 3) the gateway is not yet acting as a firewall.

I make the third assumption strictly for expediency's sake. Good security comes in layers, and it costs you nothing (other than a little setup time) to restrict where incoming VPN traffic may and may not go. Unfortunately, I can't go into much depth here; see the “FreeS/WAN quick start on firewalling” HOWTO at www.freeswan.org/doc.html for more information.

Preparing ipsec.conf

FreeS/WAN is configured via two files: /etc/ipsec.conf is its main configuration file, and /etc/ipsec.secrets is its key repository. Both files should have restrictive permissions; 0600 is a safe choice. /etc/ipsec.secrets, in particular, must be carefully protected. If you need to copy any data out of this file, say, your host's public RSA key, copy out only the data you need into a separate file. Never allow ipsec.secrets itself to leave your system. We'll discuss this file in greater depth next month.

Listing 1 shows the ipsec.conf file from an example wireless VPN client. /etc/ipsec.conf consists of three parts: basic setup parameters (config setup); default tunnel parameters (conn %default); and tunnel definitions (conn george-gracie in Listing 1, where george-gracie is the name I chose for this tunnel).

Listing 1. /etc/ipsec.conf on a Wireless Client

The default settings for the config setup section can be safely left alone on single-interfaced systems. The most important parameter there is interfaces. This tells FreeS/WAN which interface to use as the local end point of IPSec tunnels. Its default, the magic string %defaultroute, expands to ipsec0=[*interface*] (where [*interface*] is the network interface name indicated in the system's default route, e.g., eth0). Similarly, the default tunnel settings usually can be left alone.

This brings us to the heart of ipsec.conf: the tunnel definition. In the tunnel george-gracie from Listing 1, we begin by setting authby, which determines how the IPSec hosts authenticate themselves to each other. The default is “secret”, which is short for preshared secret. This setting allows you to define a secret string used in a transparent challenge-response authentication transaction. The secret itself never traverses the network, so this isn't as sloppy an authentication method as you might think.

But that doesn't matter right now, because in Listing 1 authby is set to rsasig, so that RSA authentication is used instead. RSA authentication isn't necessarily that much more secure, but it's more convenient. Whereas a shared secret must be exchanged beforehand via some secure means such as PGP e-mail or SSH, the public keys used in RSA authentication may be exchanged openly (or even published on a web page).

The idea of left and right is important in FreeS/WAN configuration; they are used in parameters and commands to designate the end points of an IPSec tunnel. Which side is which doesn't matter, so long as you're consistent. The same host should be right on both systems' tunnel definition, and one system should *not* reverse the right/left designations defined on the other.

For sanity's sake, whenever one host acts as a server to another, you're encouraged to designate that host as left. In our example, George is acting as a server for wireless clients, so George is left, and Gracie, a client system, is right.

The tunnel-configuration parameter left indicates the IP address of the tunnel's left end point, George, whose IP address is 10.0.54.2. Right, of course, specifies the right end point's IP address. In our scenario, however, Gracie and other wireless clients have dynamic IP addresses. Instead of specifying an IP, then, we'll use the magic string %defaultroute, which expands to the IP address of the interface specified in the host's default route.

The tunnel parameter leftid may seem redundant, as we just identified the left IP address with left, but it's slightly different. leftid and rightid specify each tunnel end point's authentication ID. This can be an IP address, or it can be an FQDN preceded by an @ sign. Because Gracie's IP address is dynamic, @gracie.wiremonkeys.org is the only viable value for leftid. But in this example, @george.wiremonkeys.org and 10.0.54.2 are interchangeable values for rightid.

The next tunnel parameter in Listing 1 to consider is leftsubnet. This defines which destination IPs can receive packets from the right side, and therefore for which destinations the right-hand end point will use the tunnel. Because Gracie and other wireless clients will use George as their sole gateway to the corporate LAN and to the world at large, we've set this to 0.0.0.0/0, which signifies all destinations. (There's also a rightsubnet parameter, but you need to set both subnet parameters only in site-to-site scenarios. In remote-access and other client/server setups, only one or the other is needed.)

leftrsasigkey and rightrsasigkey both are required when you've specified RSA authentication for a tunnel. The values for these are stored in each host's ipsec.secrets file, in the line beginning **#pubkey=**. Alternatively, you can use these commands:

```
ipsec showhostkey --left
```

or

```
ipsec showhostkey --left
```

The --left and --right options (which work on FreeS/WAN versions 1.9 and later) are optional but convenient. They cause the output to take the form of a leftrsasigkey or rsasigkey statement (respectively) that can be copied and pasted verbatim into ipsec.conf. For example, running **ipsec showhostkey --left** on George would return the leftrsasigkey statement in Listing 1. Running **ipsec showhostkey --right** on Gracie would return the rightrsasigkey statement. Note that although RSA keys are long, each must occupy a single line, that is, without line breaks.

The last parameter in Listing 1's tunnel statement is auto, which tells FreeS/WAN what, if anything, to do about the tunnel when IPSec is started. A value of start causes the tunnel to be initialized and started automatically. “add” causes it to be added to the IPSec dæmon's (called pluto) connection specification, and “ignore” causes the tunnel to be ignored. In Listing 1, auto is set to start. Therefore, whenever IPSec is started on Gracie, we want the tunnel to George to be brought up immediately. The config setup parameters plutoload and plutostart must be defined properly for auto to have any relevance—see the ipsec.conf(5) man page for more information.

Okay, that's it for the client-side ipsec.conf. But what about George? As it happens, in this scenario the configuration is nearly the same on both sides. Listing 2 shows most of George's /etc/ipsec.conf file.

Listing 2. /etc/ipsec.conf on Gateway

The first difference is that unlike Gracie, George has multiple interfaces, so it's necessary to give an explicit value to the interfaces parameter. The interface in George's default route faces the corporate LAN, not the wireless LAN, so a value of %defaultroute won't work. Next, we have a new config setup parameter, forwardcontrol. When set to yes, it tells IPSec to turn on IP forwarding when needed and to turn it off when IPSec is shut down.

Next, in the tunnel section itself, right is now set to %any rather than %defaultroute (because %defaultroute would return *George*'s local IP, not Gracie/right's). And, auto is set to add rather than start, because George is acting as a server; it only needs to be ready for Gracie to start the tunnel.

Starting and Testing the Tunnel

And now the moment of truth! First on George and then on Gracie, we enter the command:

```
ipsec setup restart
```

George will read /etc/ipsec.conf, load the george-gracie tunnel definition into its connection setup database and wait for connections. Gracie will do the same thing and then bring up the tunnel. Startup messages will be logged to /var/log/messages or /var/log/secure. If on the client system the output from **ipsec setup restart** ends with an “IPsec SA established” message, your tunnel is up! Try pinging or otherwise connecting to hosts on the remote network; the connection should behave no differently from before when you brought the tunnel up. In fact, you may want to run **tcpdump** on your tunnel-bound Ethernet interface to make sure that only ESP (Encapsulating Security Payload) packets (i.e., encrypted tunnel packets and not actual Ping, FTP packets, etc.) are being sent out.

Next month we'll look at another VPN scenario or two and delve deeper into the splendors of FreeS/WAN. Hopefully this was enough to get you started down the path to secure wireless networking!

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

Derivative Works

Lawrence Rosen

Issue #105, January 2003

When is one program a “derivative work” of another?

Many users of open-source software are frightened by the term “derivative works”. They worry they might accidentally create derivative works and put their own proprietary software under an open-source license. This is a complex topic that courts and lawyers disagree on, but I think we find definitions to ease people's concerns.

First, a brief reminder of why the term derivative work is so important. Here's what a typical license might say: “Licensor hereby grants you a license...to prepare derivative works based upon the original work and to distribute those derivative works *with the proviso that copies of those derivative works that you distribute shall be licensed under this License*.” See, for example, the GNU General Public License at www.gnu.org/licenses/gpl.html or the new Open Software License at www.opensource.org/licenses/osl.php.

How can you tell when you've created a derivative work? The Copyright Act, at 17 U.S.C. §101, is a little vague and doesn't say anything at all about software:

> A “derivative work” is a work based upon one or more pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation or any other form in which a work may be recast, transformed or adapted. A work consisting of editorial revisions, annotations, elaborations or other modifications which, as a whole, represent an original work of authorship, is a “derivative work”.

Almost everyone agrees that if you take the copyrighted source code of any program and physically modify it—actually revise the program or translate it into another computer language—you have created a derivative work of that program. That's the simple case. If you do such a thing with a program licensed under the GPL or the OSL, you must honor the reciprocity provision and publish the source code of the derivative works you distribute.

But what happens if you merely copy an original program as a component in your own, perhaps larger, work? Does it make a difference that you didn't actually modify the source code to combine the original program into your larger work?

Does merely linking to a program without any change to the original source code create a derivative work of that program? Almost every program links to library routines. Surely, one doesn't create a derivative work of a library simply by calling a sqrt function in the library. Why should it be any different when you link to something as complex as an enterprise server or database engine? What about linking from a software program, such as when linking your device driver into a GPL- or OSL-licensed program like Linux?

Does it matter what technical form of linking you use? Or is that analysis (e.g., static linking, dynamic linking, passing data through an API, as an object contained within a larger object, etc.) a technical morass that obscures the fundamental issue? How can the law of derivative works keep up with technological change in the world of software engineering?

These questions are important because some licenses require you to publish the source code of your portion of the resulting derivative work program, a burden you may not want to accept. Here's how I would decide in the cases described above.

1) The primary indication of whether a new program is a derivative work is whether the source code of the original program was used, modified, translated or otherwise changed in any way to create the new program. If not, then I would argue that it is not a derivative work.

2) The meaning of derivative work will not be broadened to include software created by linking to library programs that were designed and intended to be used as library programs. When a company releases a scientific subroutine library, or a library of objects, for example, people who merely use the library, unmodified, perhaps without even looking at the source code, are not thereby creating derivative works of the library.

3) Derivative works are not going to encompass plugins and device drivers that are designed to be linked from off-the-shelf, unmodified, programs. If a GPL-covered program is designed to accept separately designed plugin programs,

you don't create a derivative work by merely running such a plugin under it, even if you have to look at the source code to learn how.

4) In most cases we shouldn't care how the linkage between separate programs was technically done, unless that fact helps determine whether the creators of the programs designed them with some apparent common understanding of what a derivative work would look like. We should consider subtle market-based factors as indicators of intent, such as whether the resulting program is being sold as an “enhanced” version of the original, or whether the original was designed and advertised to be improvable “like a library”.

You should care about this issue to encourage that free and open-source software be created without scaring proprietary software users away. We need to make sure that companies know, with some degree of certainty, when they've created a derivative work and when they haven't.

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

email: lrosen@rosenlaw.com

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

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #105, January 2003

Readers sound off.

Possible Attack on "Stealth" Log Host?

Thanks for "Stealthful Sniffing, Intrusion Detection and Logging" [*LJ*, October 2002, also available at www.linuxjournal.com/article/6222]. That is a clever use of Snort, particularly as a means to send logs to a remote host. Just a couple of comments as I read through the article. In the Sidebar titled "LAN Segments, Hubs and Switches" on page 38, it is not impossible to send logs to the Snort logging host from a distant segment. Just ensure the Snort host is attached to a valid IP subnet, and give the subnet's router a static ARP entry that associates the Snort host's MAC address to an unused IP address that is valid for the Snort host's subnet. With the method of logging outlined in the article, it may still be possible to compromise the integrity of the Snort logging host. An attacker may be able to send their own log messages, which may be used either to overflow the logging disk or to attack a potential vulnerability in Snort's capture/parsing logic (if there ever was one).

—John Kristoff

Mick Bauer replies: Even a stealth logger is vulnerable to denial-of-service (DoS) attacks and sniffer-specific application exploits. But in practical terms, neither of these strikes me as a huge risk. I think it would be pointless for an attacker to aim specifically to take out a stealth logger via a DoS attack; that amount of traffic probably would also take out the system you're actually trying to attack without being logged. On the other hand, we should keep in mind that the "stealth sniffing" technique is intended to mitigate the specific threat of attacks against the logger's local TCP/IP stack and applications. I do think it's useful for that and effectively raises the bar considerably against would-be log-tamperers. However, these are good observations.

More C++ Coverage, Please

I really enjoyed the article “Memory Leak Detection in Embedded Systems” that appeared in the September 2002 issue of *Linux Journal*. Lately I have been doing a lot of C++ development with the DB2 UDB Administrative API, and detecting memory leaks in both my code and IBM's is useful for debugging purposes. Please feature more articles that address these types of C++ development topics in 2003. An overview of compiler and compiler tools (g++, Intel compilers, etc.) tools would be one article in particular I would like to see run.

—Kevin Wittmer, Senior Software Developer, Expand Beyond, Chicago, Illinois

Watch for an article on the Intel C and C++ compiler next issue.

—Editor

Red Hat 8.0: Love the License, Hate the Look

I recently purchased the new Red Hat 8.0. I have been following Red Hat since 5.1 and have purchased 6.0, 6.1, 7.2 and downloaded 7.3. I find the new desktop, Bluecurve, to be a true bastardization of both Gnome and KDE. Red Hat has removed all of the character from Gnome and KDE and smudged the windowing into their own proprietary look and feel. Konqueror is not the default browser in the KDE menu, and CUPS is not installed by default. If I want to go into KDE, I want KDE. If I want Gnome, give me Gnome, not some smudging of the two. On the other hand, the beauty of open source is that you can do what you want with it as long as you publish the source code, and that is what Red Hat has done. Good for them. But, Bluecurve should be called BLUE-SMUDGE.

—Tom Amon, IT Tech

More on Getting Started?

I love the magazine and enjoy it and sharing GNU/Linux with everyone I know. When I got started with GNU/Linux I looked to books and mags to help me, and what really helped me were any tidbits of information that I could get. Marcel's sections are a blast to read, and maybe something like that more directed to newbies would help other new people in their exploration of GNU/Linux.

—Matthew

If you enjoy sharing your knowledge, please visit our web site (www.linuxjournal.com) for an author's guide, and send us an article proposal.

—Editor

Keyboard Changes?!?!

The key for the EXCLAMATION MARK (!) should be shifted to the key left of the number one (1). This will facilitate typing of the EXCLAMATION MARK (!) without using the Shift key. This is necessary because EXCLAMATION MARK (!) is used very often while typing any text. The key for the QUESTION MARK (?) should be brought down below on the same key on which it is available presently, so that Shift key need not be pressed. The OBLIQUE symbol (/) is very rarely required. Lastly, the INDICATOR LIGHT ON for the CAPS LOCK key, pressed for typing ALL CAPS, is not enough to highlight the same. The INDICATOR LIGHT should not be constant. Instead, it should go on blinking, preferably with a “BEEP BEEP” sound so as to attract the attention of the USER, so that he undoes it after his need to type ALL CAPS is over. I hope, the above-mentioned suggestions are worthy of due consideration and if possible, implementation.

—DR. C.H. AWALGAONKAR

See **man xmodmap** for how to remap any key to any character.

—Editor

Picking a Linux SAN?

I'm a regular reader of your magazine here in the UK. I would really like to see a good, well-researched story on IP SANs, iSCSI and Linux. I've been noticing lots of activities in this area, and it looks like Linux is playing a big role in this. I got involved in the subject because I'm on a committee that should (supposedly) decide on network storage strategies at our place (News International, big corporation). There are a couple of small companies already offering iSCSI targets, and all of them appear to be Linux-based. See for instance www.pyxtechnologies.com (this is Andre Hedrick) or technomagesinc.com. The subject is pretty hairy. If you want more information or links, let me know.

—Antonio Cordova, Pre-Press Consultant, News International

What Are BogoMips?

I saw, in the September 2002 issue of *LJ* [“What Has 1.1 Terabytes, 9,503 BogoMips and Flies?”], a project system that listed the CPU performance in BogoMips. What is a BogoMip, and how is it measured?

—Frank Jansen

BogoMips are a measurement of how fast the kernel runs a simple delay loop. See the BogoMips Mini-HOWTO at www.tldp.org/HOWTO/mini/BogoMips.html.

—Editor

Font Question

What font are you now using in the headlines on the cover of your magazine? I know this is a strange question, but I rather like that font, and being a graphic designer, I would like to use it.

—Tiago Oliveira

Lydia Kinata replies: We use the DaxWide font family on the cover and for article titles and headers within the magazine. DaxWide comes in several weights, from very light to “Extra Black”. It's a very useful font.

Best Practices for Web Security

I read your article on secure PHP applications in *Linux Journal*, October 2002, and I thought it was quite useful. An additional source of information on the subject is the “The OWASP Guide to Building Secure Web Applications and Web Services” (www.owasp.org/guide). A lot of “good practices” are combined in a single document.

—Peter Fokker

An Ad Is an Ad

I know some people think the “battle” between Linux and Microsoft is a religious quest, but the letter from Renato Carrara is just plain silly. [See the “Don't Run Microsoft Ads” letter in *LJ*, November 2002]. A magazine is a business proposition, and ads pay for the magazine. As long as they don't flagrantly violate good taste, magazines should run whatever ads are proffered. Microsoft is probably astute enough to understand it would not find a receptive market here.

—Gary W. Nickerson, Director, Information Technology, Rockefeller Brothers Fund

Erratum

In the November 2002 issue, “2002 Readers' Choice Awards” (page 72), the number of voters given in the first paragraph is incorrect. It should read “Almost 6,000 voters...”.

Advanced search

UpFront

Various

Issue #105, January 2003

LJ Index and more...

diff -u: What's New in Kernel Development

At long last, **Jeff Dike's User-Mode Linux** has been accepted into the 2.5 kernel tree. Now users may invoke the Linux kernel as a user process on a running system. Or to put it another way, users may invoke any number of concurrent virtual Linux systems, which may be Beowulfed together or clustered in some other way, or else used for testing new drivers and other invasive patches that can result in crashed systems. No more lengthy reboots when your new toy blows up—start another instance and keep hacking!

SGI's journaling filesystem **XFS** has also been accepted into the 2.5 tree, having finally met **Linus Torvalds**' requirements. For a long time, XFS made some fairly violent changes to areas of the kernel that Linus felt really shouldn't be touched. These have been cleaned up, and the code is in. Users desiring journaling filesystems can now choose between ext3, ReiserFS, IBM's JFS and now XFS.

Threading scalability has just made a huge jump. Now you can start 100,000 threads and run them all concurrently. This came as such a shock to kernel developers that Linus thought he misread the announcement. **Ulrich Drepper** and **Ingo Molnar** have been leading this charge, which brings threading scalability far, far above what anyone could reasonably need on a home system. Still, as one developer put it, if nothing else, this shows we're doing something right.

There is currently an ongoing debate that will have a strong impact on virtually all users. The ability to unload a given module from the kernel may be going away. Apparently, the code to handle **modules** has become too complicated; one proposal is that this can be simplified if modules, once loaded, are simply grafted permanently onto the running kernel. The problem with this is the ability to unload modules is seen as a really useful feature many developers want to keep. So far, the debate looks as though it could swing either way.

Some new code for handling crashes has come to light. One such is a patch to control **core-dump** filenames. When a program core-dumps, instead of simply producing a file in the current directory called core, it can produce a file in another directory with a name specified by the user. Another bit of code for crash handling is **kksymoops**, an OOPS handler that decodes the kernel symbols before dumping the OOPS. Previously, OOPS output had to be run through a separate program in order to produce data that would be meaningful to kernel developers. With kksymoops, more and better information is available. This makes it easier for users to report system crashes and easier for developers to debug them.

—Zack Brown

CheckBook Tracker tony.maro.net

If you've been waiting for a replacement for Quicken, this might be the ticket. I don't have Quicken, so I downloaded my bank files as QIF, and they imported flawlessly into CheckBook Tracker. The interface is clean, efficient and has a number of display modes. This is probably the most intuitive checkbook I've used recently. Requires: libXi, libXext, libX11, libm, libgtk, libgdk, libglib, libgdk_pixbuf, libgmodule, libdl, glibc.

—David A. Bandel

TypeFast homepages.ihug.co.nz/~syringe/typefast.html

This is a simple curses-based typing practice program. It won't teach you to type (you already need to know how to do that), but it will help improve your speed, using either an xterm window or a VT. You can practice on different keyboards (QWERTY or DVORAK) and alternate hands to give either hand more practice. Requires: libncurses, glibc.

—David A. Bandel

LJ Index—January 2003

1. Number of Linux-based Sicom Systems point-of-sale systems being installed in Burger Kings in Puerto Rico: 160
2. Number of Linux-based PDAs: 16
3. Billions of searches per month on Linux-based Google: 5

4. Millions of Linux users: 18
5. Position of Linux Professional Institute (LPI) on *Computer Reseller News* (*CRN*) fastest-rising certification list: 1
6. Position of Red Hat Certified Engineer (RHCE) on *Computer Reseller News* (*CRN*) fastest-rising certification list: 2
7. Percentage of retail Red Hat Linux customers who use the OS at home: 34
8. Percentage of retail Red Hat Linux customers who use the OS at work: 13
9. Percentage of retail Red Hat Linux customers who use the OS both at home and at work: 50
10. 2001 salary in dollars of Hilary Rosen, president of the RIAA: 1,163,729
11. 2001 salary in dollars of Jack Valenti, president of the MPAA: 1,030,000
12. Average payola in dollars paid by record companies to US commercial radio stations to add a song to a playlist: 1,000
13. Payola in dollars paid per week by the US record industry: 300,000
14. Amount paid by US radio stations to record companies or artists to play music: 0
15. Range of percentage of revenues that will be paid by small webcasters to artists via the RIAA's collection arm: 8-12
16. Percentage of TV commercials that don't get watched in households using the Linux-based TiVo: 88
17. Factor by which the cost of processing a single bit declines every 20 years: 1,000

Sources

1. 1,2: LinuxDevices.com
2. 3: Blogads
3. 4: Linux Counter
4. 5, 6: *Computer Reseller News*
5. 7-9: InternetNews
6. 10, 11: *Washington Business Forward*
7. 12, 13: *Salon*
8. 14, 15: MediaPost
9. 16, 17: Martha Rogers, Peppers & Rogers Group

(Linux + Java) - Bulk = Kaii

Infomart's Kaii (www.kaii.info) has joined Sharp's Zaurus in the growing field of Linux-based PDAs. Like Zaurus, Kaii will run Lineo's Embedix Plus Linux kernel v. 2.4.2, Trolltech's Qt/Embedded and Qtopia environment and Insignia's Java

Virtual Machine (JVM). Unlike Zaurus, it runs on a 160MHz Hitachi SH3 CPU (Zaurus runs on a StrongARM) and uses an on-screen keyboard.

Kaii comes with a choice of color or monochrome 320 × 240 screens, up to 128MB RAM, 32MB masked ROM or Flash, USB interface, RS-232C serial port, IrDA port, CompactFlash Type II slot and an MMC slot.

It also comes with the Opera browser, the Hancom Mobile Office suite and sync software for Linux desktops, OS X, Windows and "PIMs like Outlook".

Its size is 137 × 73 × 17mm or 5.39 × 2.87 × 0.67in, which is nearly identical to the Zaurus without its keyboard exposed.

Infomart is positioning Kaii as a "low-cost alternative computer or corporate special purpose workstation in cost-sensitive markets like India, China, Eastern Europe, Africa, etc." Infomart is headquartered in India.

—Doc Searls

They Said It

Programs that use treacherous computing will continually download new authorization rules through the Internet, and impose those rules automatically on your work. If Microsoft, or the US government, does not like what you said in a document you wrote, they could post new instructions telling all computers to refuse to let anyone read that document. Each computer would obey when it downloads the new instructions. Your writing would be subject to 1984-style retroactive erasure.

—Richard Stallman, on why "trusted" computing shouldn't be

The record companies hold all the cards; if you want to be famous, you have to go the mainstream route. If you want huge success, you have to go the mainstream route. If you want worldwide success, you have to go the mainstream route. And until we see our first Internet & Live Shows Only artist sell a million CDs without a label deal, the major labels will be the only mainstream route available. Don't quote Grateful Dead statistics to me—they're the exception, not the rule.

—Janis Ian

This page was generated entirely by computer algorithms without human editors. No humans were harmed or even used in the creation of this page.

—Google News disclaimer

I think ultimately we will look for an open-source desktop. I think that's eventually where the industry will go.

—Richard Thwaite, director of IT and ebusiness infrastructure for Ford Europe (which controls 33,000 desktops)

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

—Lawrence Lessig

Useless?

Don Marti

Issue #105, January 2003

ping6 is the new ping.

In the early 1980s, I had a 300 baud modem. From a business point of view, it was just about useless. Our local BBS sysop, Greg Corson of “The Connection”, offered turnkey BBS systems to businesses, so that employees could have mail, discussion and file transfer, but he was way ahead of his time. Businesses took a pointless detour to fax machines before they understood the power of offering BBS-like services to everyone.

Like that old modem, all new networking technologies look useless—until enough people get on them. Then, well, it's called a “network effect” for a reason.

So what's the “useless” technology to play with today? IPv6, the next version of the Internet Protocol. You won't find news and business sites using it; you can't send most people mail on it; you can't call up the phone or cable company and get it, and if you put up an IPv6 web site, Google won't crawl it. On the other hand, IPv6 has enough unique IP addresses (2128 of them) that no matter what kind of internet service you have, you can have a static address of your very own.

You can really get that address and do something with it, too. Viagénie, a Canadian company, sponsors freenet6.net, an easy-to-use service to tunnel IPv6 over your existing net connection. I worked through Peter Todd's article on page 64, and it took me less time to finish and “ping6” his IPv6 box, than to read the article. And I read fast.

You can do more with IPv6 than just ping Peter's box—Ibrahim Haddad and David Gordon take a look at how well Apache handles IPv6 traffic on page 86. And once you're on IPv6 with a static address that's all yours, you can start to learn more.

Another great network project is a VPN to link sites securely over an existing connection. Be a hero at work when you replace an expensive leased line or proprietary VPN box with FreeS/WAN, using Mick Bauer's instructions on page 30. How much did you pay for *Linux Journal* again? We're such a bargain.

Software you write probably calls read() and write() to get data from disk and send it to the network, or vice versa. But behind the scenes, the kernel is copying the data for you. Can you speed things up? Dragan Stancevic takes you along the path to zero copies and explains the sendfile() system call along the way, on page 48.

Guylhem Aznar has written two articles for our web site on hot new apps for the Linux-based Zaurus PDA, and this month he's on page 38 with recommended software, and some key hardware, to make your Zaurus (you *do* have one, right?) more useful than ever.

There's plenty of other good stuff this month too. Java doesn't have to be just a bytecode language—you can compile it like C (page 74). And when I make an SSH connection across a flaky network, I want to be able to pick up where I got thrown off. The **screen** program lets you do that and more, on page 80.

Finally, the cover. Robin Rowe has written us another Linux movie article, this one about the latest *Star Trek* movie. *Linux Journal* doesn't get to see the Linux movies before everyone else, but do you really think Starfleet gets by with only 232 IP addresses?

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

On the Web

Heather Mead

Issue #105, January 2003

Can anyone rely on the government to be at the forefront of technological innovations?

The theme of our November 2002 issue was internationalization, and we were fortunate to receive quite a few interesting articles on the subject. One story in particular, Wayne Marshall's “Radio E-Mail in West Africa”, gave insight into how much we can accomplish when we are willing to shift our perspectives a bit and work with the tools available—in Wayne's case, HF radios capable of long ranges but little bandwidth. Unfortunately, space constraints in our print edition forced us to cut some of the details of the e-mail configuration used from the article. The long version, however, is available on our web site at www.linuxjournal.com/article/6299. If you're interested in the details of how Wayne's radio e-mail project came about or how the setup differs for incoming as opposed to outgoing mail, be sure to read “Radio E-Mail in West Africa: the Complete Version”. And be sure to check out the readers' comments at the end; a lot of people are sharing their unique communication setups.

On the topic of internationalization, Fred Noronha has been providing us with Linux and open-source news from India and neighboring countries. The grassroots support for software libre is strong, and even the government has expressed interest in moving their systems to open source. But are they serious? In a follow-up web article, “Indian Government's Reported Move Makes News, Then Fuels Skepticism” (www.linuxjournal.com/article/6389), Fred reports that many speculate the Indian government is simply using free software as a threat to get Microsoft to lower license fees or make a local investment. As one reader states, India cannot rely on the government to lead the Free Software movement. “So we the people will adopt free software and eventually the government will have no choice but [to] follow the people.”

Finally, we're certainly not opposed to stirring up some trouble; we just don't always know where it's going to come from. So quite innocently we announced the winners of the 2002 Readers' Choice Awards on our web site in mid-November in the form of a press release (www.linuxjournal.com/article/6380). We certainly weren't expecting the amount of reader comments we received—42 comments, to be exact, as I write this. So who comments on a press release? People who agree on one thing: our voters/readers are idiots because they didn't choose the *correct* winners. Some days it feels like we are all back in the sandbox, ready to hit each other on the head with our toy trucks if we don't get our way. So jump on to the article's web page and throw some sand yourself.

If you want to share the details of a unique system you've built at home or far away, drop us a line at info@linuxjournal.com. Remember to check the *Linux Journal* web site often; new articles are posted daily.

Heather Mead is senior editor of *Linux Journal*.

Best of Tech Support

Various

Issue #105, January 2003

Our experts answer your technical questions.

Bulletproof Backup /boot?

I have a number of remote Linux workstations that I need to ensure are always available to me remotely. I am looking to set up these systems in a TiVo-like fashion, whereby each machine would have two boot partitions and two root partitions. The goal is to be able to perform maintenance operations on the primary-use partitions via a secondary-use boot and root partition. I require only basic functionality while in the secondary partition, namely e2fsprogs and a DHCP-networked sshd. I intend to use LILO or Grub and an init6 to switch between the setups. Assuming that console access is never an option, what is the most bulletproof way to lay out the partition tables and configuration for something like this?

—Marc Lavergne, mlavergne@cfl.rr.com

You really don't need a second partition to boot in to for updates. RPMs allow updates on the fly. In fact, the only reason you need to reboot is to run an upgraded kernel. That being said, the ping-pong methodology you discuss is always a good idea if you mess something up during an upgrade. AFAIK, LILO and Grub do not support a feature that allows you to boot an alternate partition if the previous boot failed. You would probably need to write some code to accomplish this. This solution, however, will not fix system hangups or hard crashes. Truly remote systems usually need some hardware console support.

—Christopher Wingert, cwingert@cwingert.qualcomm.com

It sounds as though you need a solution that doesn't require the remote user's assistance. Otherwise, I would recommend a bootable CD; some, like Gentoo, even include SSH functionality.

—Chad Robinson, crobinson@rfgonline.com

There's not really one partition table setup that would be better than the others. The only thing that's a tad more bulletproof is to use only four partitions so you don't use extended partitions. That way, your entire partition table is in the master boot record of your disk and not spread out around your disk in a linked list. That said, it won't really make a huge difference, unless very bad things happen to your disk.

—Marc Merlin, marc_bts@google.com

Linux Driver for Belkin Wireless Card?

I recently installed a wireless PCI network card, but the company, Belkin, doesn't supply a Linux driver. I searched around and found the Linux-WLAN driver. I tried to install it, but when I was making the driver (**make all**), I received several error messages, and it wouldn't compile.

—Matt Jacobs, mjacobs519@hotmail.com

If you want to get on the Net fast, the good news is Belkin seems to be using the common Prism/2 chipset, like many other vendors. Tim Miller has put together an RPM for Red Hat 7.3 and 8.0 to support Prism/2-based cards (prism2.unixguru.raleigh.nc.us). If you want to build drivers from source, either because you want a custom kernel or want to learn how it's done, work through the Kernel HOWTO (www.tldp.org/HOWTO/Kernel-HOWTO.html). Make sure you have everything needed to build a “stock” kernel first.

—Don Marti, info@linuxjournal.com

CD Install Hangs

I tried a number of times, by booting from the CD-ROM, to install Red Hat 7.3 in a NEC P150 machine. But I keep getting hung up half-way through the installation.

—Leong Y C, leongyc@tp.edu.sg

Try changing the use of DMA on the BIOS for your CD-ROM drive unit. On some machines the CD drive hangs if this is set incorrectly. This can happen not only with Linux, but with any operating system and with applications on CD.

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

You can try three things: 1) disable autoprobing of hardware (boot with **noprobe**); 2) try an install in text mode; and 3) if you purchased it, talk to Red Hat for installation support.

—Marc Merlin, marc_bts@google.com

Making Snort, MySQL and ACID Work Together

Does anybody know where I can find a one-stop shop that shows in-depth how to install Snort on a system with MySQL and ACID? I have been trying this for three weeks, but the ACID console will not log the events.

—Colin Slevin, Colslev@transwareplc.com

You don't mention whether entries are being logged via other means. Snort typically will log to a number of channels in its default installation. If you aren't getting entries logged at all, the problem is likely elsewhere. Otherwise, edit your snort.conf file. A number of logging examples are there, and you should be able to mimic one that suits your needs. Snort has a mailing list whose users are very supportive. You can subscribe by visiting www.snort.org/lists.html. I recommend submitting your question there. Supply your exact installation procedure so somebody can help you more thoroughly.

—Chad Robinson, crobinson@rfgonline.com

Sharing a Windows Box's Printer

How do I use a shared printer connected to a Windows 2000 Pro PC? I have Mandrake 8.2.

—Sam Dorrough, dorroughs@microdyne.com

Check that the printer is shared from the Windows box and that you have Samba installed on the Linux box. Then run **printtool** on the Linux box to set up access to the printer.

—Christopher Wingert, cwingert@cwingert.qualcomm.com

It is best to share the printer with a Windows box that has a name of eight characters or less. I have had problems with longer names of shared devices. Documentation on Samba and printers can be found at: us1.samba.org/samba/docs/Samba-HOWTO-Collection.html.

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

New Products

Heather Mead

Issue #105, January 2003

Runtime Revoultion 2.0, Nuxeo Collaborative Portal Server, CommuniGate Pro Groupware and more.

Runtime Revolution 2.0

Runtime Revolution announced the release of Revolution 2.0, a multiplatform development tool. Revolution uses an interface builder, an English-like programming language and built-in internet and multimedia features, so users can write a single application that will run on any target OS. Features new to version 2.0 include a Jaguar-friendly interface, Unicode text entry and manipulation, spreadsheet/table text fields, a new XML library, new SOAP support, a new report printing engine, an integrated debugger including scripting support, improved database access on all platforms, Perl-compatible regular expressions, and MIDI music file-creation and playing tools.

Contact: Runtime Revolution Ltd., 91 Hanover Street, Edinburgh, Scotland, UK EH2 1DJ, support@runrev.com, www.runrev.com/news/revolution20.html.

Nuxeo Collaborative Portal Server

Nuxeo Collaborative Portal Server (CPS) is a web content management system (CMS) implemented on top of Zope. CPS can be used to implement collaborative intranet, extranet or internet applications. In CPS, the generation of web pages and hierarchy structures are dynamic and rely on a separation of content, presentation of data and logic that applies it. Indexation and previewing of office automation documents in a variety of formats (StarOffice, OpenOffice, PDF, RTF, XML, etc.), an integrated search engine and meta directory, versioning and mailing lists are all available. CPS is free software that can be downloaded from the Nuxeo web site.

Contacti: Nuxeo 14, rue du Soleillet, Paris, France 75020, contact@nuxeo.com, www.nuxeo.com/en.

CommuniGate Pro Groupware

CommuniGate Pro Groupware messaging server offers e-mail, calendaring, scheduling and sharing capabilities. Targeted as enterprise-level use, the Groupware product can replace or run in conjunction with Microsoft Exchange and offers more stable and secure functionality. Groupware operates on 26 OSes, allowing them all to connect to one another through a message store and transport MAPI provider that composes and parses MIME messages. CommuniGate Pro Groupware supports SMTP, POP, IMAP, folder sharing, webmail, LDAP, S/MIME, mailing lists and spam/virus protection for connected clients.

Contact: Stalker Software, Inc., 655 Redwood Highway, Suite 275, Mill Valley, California 94941, 800-262-4722 (toll-free), info@stalker.com, www.stalker.com.

QuickStart Kits

Ampro Computers and TimeSys have partnered to produce a series of certified, production-quality Linux distributions and development environments for the design and deployment of embedded real-time OEM devices. The first joint release, called a QuickStart Kit (QSK), is for Ampro's PowerPC ISA-based EnCore PP1 module. QSKs for MIPS and x86 boards and modules will follow. Each QuickStart Kit comes with a TimeSys Linux distribution with the OS, device drivers, GNU development tools, Windows (Cygwin) and Linux workstation development environments, a root filesystem (RFS) and a verified boot loader. In addition, QuickStart Kits include evaluation copies of Loadable Kernel Modules (LKMs) to add the real-time reservations capabilities of TimeSys Linux Real-Time, TimeSys Linux CPU and/or TimeSys Net.

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

Xandros Desktop

Xandros has released the Xandros Desktop, a desktop environment developed by the former Corel Linux Business division team, which Xandros acquired in 2001. The main features of Xandros are compatibility, ease of use, high security and network updates. Xandros is interoperable with Windows, UNIX and Linux networks. File sharing between these systems also is possible. Desktop also features CodeWeaver's CrossOver technology so users can run Windows and Linux applications simultaneously. In addition, the five-step graphical installation program allows users to resize NTFS partitions.

Contact: Xandros, Inc., 41 East 11th Street, 11th Floor, New York, New York 10003, sales@xandros.com, www.xandros.com.

7160 Anything I/O Card

The 7160 is a standalone (no bus) version of the FPGA-based Anything I/O card series. It provides 96 I/O bits, and on four 50 pin connectors, 24 bits per connector. The connectors have I/O module rack-compatible pinouts. The 7160 also has two RS-485 serial ports and two RS-232 ports, one of which is used for downloading initial configurations to the on-card Flash EEPROM. Many I/O configuration files are provided, including both simple and smart remote I/O, 4- and 8-axis servo motion control and more. Available I/O interface daughter cards include an 8-channel RS-232/485 interface, a debug LED card, a 2-axis #A stepper motor driver and more.

Contact: MESA Electronics, 4175 Lakeside Drive, Suite 100, Richmond, California 94806, sales@mesanet.com, www.mesanet.com.

SuSE Openexchange Server

SuSE Linux announced the Openexchange Server, an all-in-one communications and groupware solution. Openexchange provides e-mail functions, web-mail clients, a central appointment and address management system, as well as project management and task planning tools in one application. A centrally controlled document management tool and group-based discussion forums are also supported. The Openexchange interface runs on common browsers, so users can access the services on all computer platforms, including Windows and Mac OS. The data synchronization feature allows information to be shared with PDAs.

Contact: SuSE Linux AG, Deutschherrnstr. 15-19, Nürnberg, Germany D-90429, presales@suse.de, www.suse.de/en.